Technical notes provide unique applications, innovative methods, and clear protocols designed specifically for Stratagene reagents, instruments, and software.

Technical Note

Stratagene's ArrayAssist® software packages are designed for analysis of microarray results using a variety of different algorithms to discern biologically relevant, expression-level changes from a very large number of data points. The software uses analytical features such as multi-way ANOVA, non-parametric statistical tests, classifiers for machine learning-based prediction, and sophisticated and interactive visualization options. The software also includes functionality to run custom analysis scripts (using R-scripts) to support those researchers who want to look at their data in ways that differ from those in the ArrayAssist® Software graphic user interface.

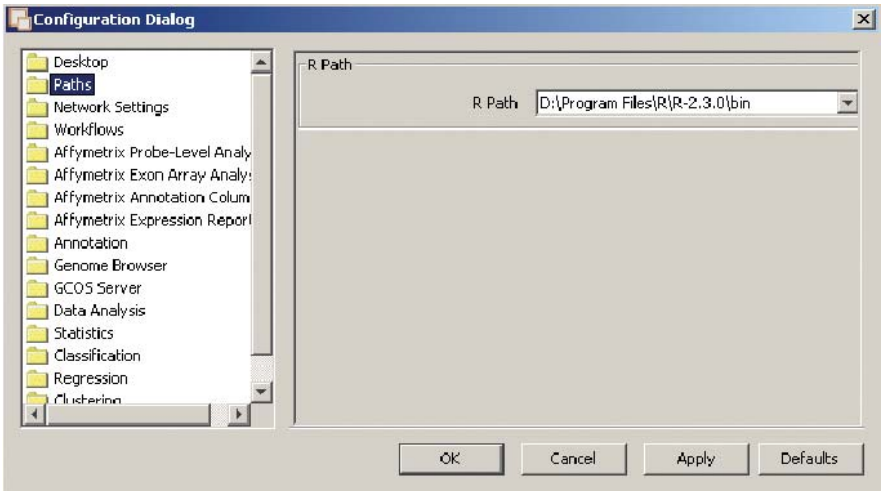
Introduction

R is a language and software environment for statistical computing and displaying graphics for data analysis. The ArrayAssist® Software has the capability to run R-scripts on the data loaded into it, and import the results of running the R-script back into the ArrayAssist software. Users are required to independently install and set up R on their machine to create the R environment to be used. This Technical Note outlines how to set up the R environment and provides instruction on running R-scripts and importing the results back into the ArrayAssist software. It also provides sample R-scripts that can be run from the software.

Protocol:

Setting up the R environment:

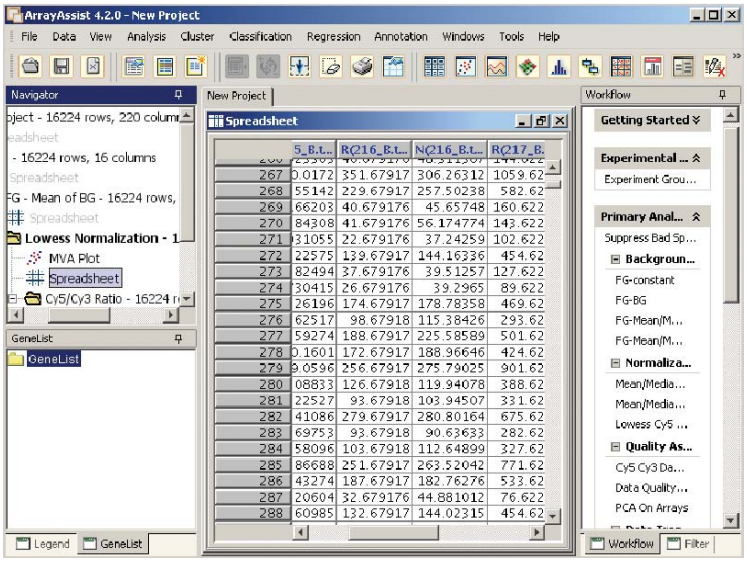
- Download the R package from http://www.r-project.org for your particular platform: Windows®, Linux, or Mac®.
- Install the R package by following the included instructions.
- Install other R libraries and modules that you may need. You may be required to install the Bioconductor package if you wish to use R for microarray summarizations and normalization packages that are included in R.
- You will need to set the directory path to the R executable.
    + On the Windows platform, you have to include the directory path to the R executable in your Path Environment Variables. To do this, go to My Computer, right-click, and go to the Properties dialog. Go to the Advanced tab. Click on the Environment Variables. In the System Variables list, choose Path (under "Variable") and add the directory path to the R executable in the "Values" field. Note there may be other directory paths already present in the Values field.
    + On the Linux platform, check to see if R is included in the path variable by typing echo $PATH. If the path to the R executable is not included, include the path to the R executable in the .bashrc by including the line:
      export PATH=$PATH:<full_path_to_R_executable>
    + If you do not want to include the path to the R executable in the PATH variable of your system, you can enter the path to the R executable in the ArrayAssist software by navigating the R path in Tools>Options>Paths>R Path. Note that you only need to set the path to the R executable once, either in the system path or setting up the path to the R executable in the ArrayAssist software.



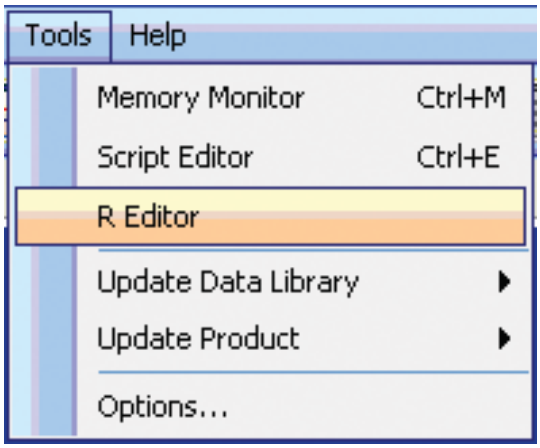
Specifying the R path directly within the ArrayAssist software by using the Tools>Options drop-down menu.

Calling R-script from the ArrayAssist® Software:

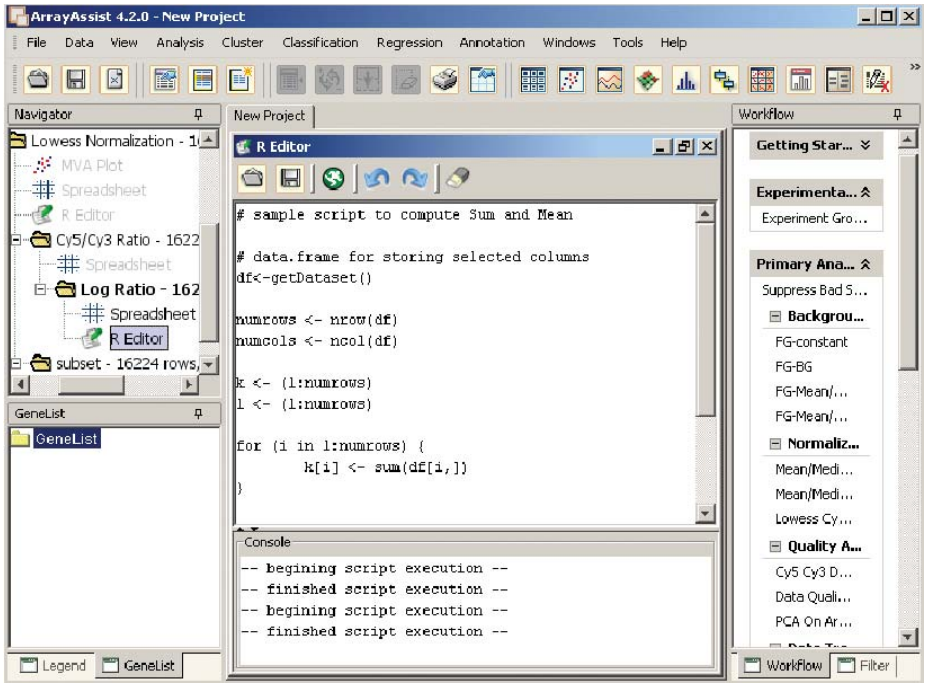
- First verify that your installation is properly configured and that you are able to call R from the ArrayAssist software.
- To call R, launch the ArrayAssist software and load a dataset.



- Open the R-script editor from Tools --> R editor. This will launch a script editor.

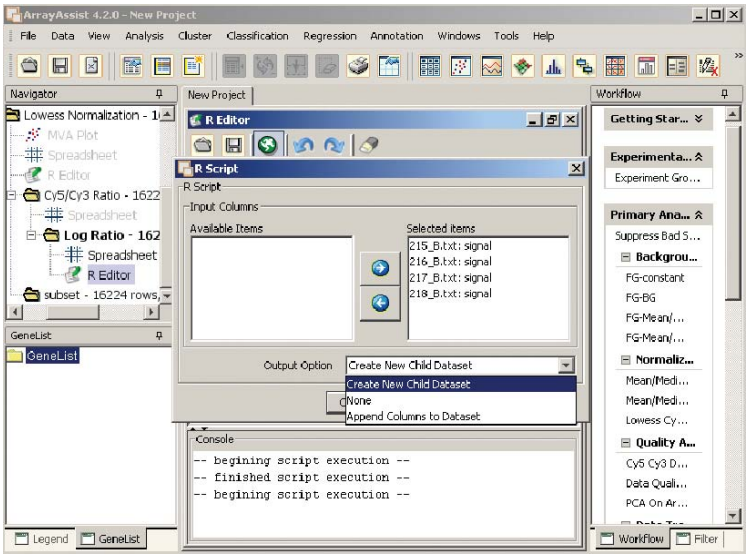


- The R editor has two panes. The top pane is the area to write and load the R-script, and the bottom pane is the Console, where the message and output from R are posted.

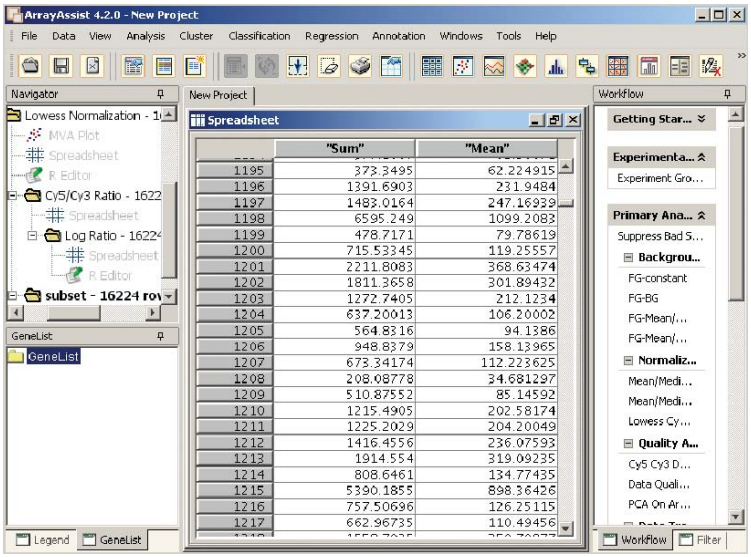


- Click on the file open icon on the R editor. Navigate to the <arrayassist_install_dir>/samples/rscripts and load a sample script. The sample script will be loaded into the R editor. Examine the script to ensure it will perform the analysis in the way you want. The structure of the R-script and examples of how scripts are set up will be discussed below.

- Click the Run icon, the third icon on the R editor. Running this sample R-script will pop-up a column chooser dialog with the columns in the dataset. You can choose the columns of the dataset that you want to send to R. These are the columns on which the R-script will operate. Select the columns that you want and click on the arrow to move the required columns to the Selected Items box. You will see an Output Option field at the bottom of the dialog. In your script, you may have results that are from each row of the dataset. This would be a column vector. Or you may just have some summarized value or values. If you have columns as output of the R-script, you may want to choose the options Append Columns to Dataset, or Create New Child Dataset in this field. If you choose to create a new child dataset, this will be displayed as a new data object in the navigation tree. The Output Options field also includes the ability to print the results to the console or create a table from the output of R. Choose the appropriate option from the drop-down list.



- Observe the output of the script in the Console. If the script runs successfully, a message will indicate this.



Troubleshooting:

If running the sample script is successful, you have everything set up on your machine to run R-scripts from the ArrayAssist software.

If the script was not successful, the R executable most likely was not accessible from the ArrayAssist software. You will need to examine if R can be run independently.

If you can launch R independent of the ArrayAssist software, then examine whether you have set up the PATH variable properly in your system. Remember to exit the software before setting the R PATH in the system. If you navigate the path variable in the Tools --> Options dialog of the ArrayAssist software as detailed in the section above, hit "Enter" from your keyboard. Note that you only need to do one of the above, i.e. setting up the path to the R executable in the system path or setting up the path to the R executable in the software.

Writing and running R-scripts in the ArrayAssist® Software:

Included here are a few example R-scripts to get you started with the R scripting utility available in the ArrayAssist software. Detailed scripting documentation which will expose all functions of the product will be released in a separate document. Utility and example scripts from the ArrayAssist software development team as well as from other ArrayAssist software users can be found at softwaresolutions.stratagene.com.

All example scripts have three parts:

- **Dataset Access**: The function getDataset(), is used to access the active dataset.
- **Parameter Passing to R functions**: For this section, you will need some knowledge of the R scripting. See http://cran.r-project.org/doc/manuals. Example R-scripts in the samples folder of the ArrayAssist software install directory can also serve as a good starting point to learn R scripting.
- **Parsing the output**: The function addResultColumns() is used to fetch back the results. For parsing the output, you will get two options. The output can append new columns to the active dataset or make a new child dataset.

Example Scripts:

Note: The scripts provided here can be copied and pasted into the Script editor and run.

Script 1:

```
#################################################################
# sample script to get clusters for the data using kmeans

# data.frame for storing selected columns; this data.frame
# is made by the columns selected by the user when this
# script is executed

df <- getDataset()

# kmeans: Perform k-means clustering on a data matrix.
#
# Usage:
# kmeans(x, centers, iter.max = 10, nstart = 1,
# algorithm = c("Hartigan-Wong", #"Lloyd", "Forgy", "MacQueen"))

# get 3 clusters for the data.frame,
# by default the algorithm is "Hartigan-Wong"

clus <- kmeans(df, 3, 50)

# add result as a column to the dataset
# columns should be added as data.frame

addResultColumns(data.frame(clus$cluster), c("Clusters"))
#################################################################
```

Script 2:

```
#################################################################
# sample script to compute Sum and Mean

# data.frame for storing selected columns
df<-getDataset()

numrows <- nrow(df)
numcols <- ncol(df)

k <- (1:numrows)
l <- (1:numrows)

for (i in 1:numrows) {
k[i] <- sum(df[i,])
}

for (i in 1:numrows) {
l[i] <- (sum(df[i,])/numcols)
}

# add result columns as data.frame

addResultColumns(data.frame(k,l), c("Sum", "Mean"))
#################################################################
```

Script 3:

```
#################################################################
# Shapiro-Wilk Normality Test
# R function is shapiro.test(x)
# x is a numeric vector of data values, the number of which must be
# between 3 and 5000.
# Missing values are allowed

# data.frame for storing selected columns

df<-getDataset()
numrows<- nrow(df)
numcols<- ncol(df)
z<-(1:numrows)
for(i in 1:numrows){

k1<-c(df[i,])
k2<-as.numeric(k1)
z[i]<-shapiro.test(k2)$p.value
}
# add result column as data.frame
# append a column to the active dataset or create a new subset of data

addResultColumns(data.frame(z),c("P-value"))
#################################################################
```

Script 4:

```
#################################################################
# unpaired two sample T-test

# data.frame for storing selected columns

df<-getDataset()

# number of rows in your selected dataset

numrows<- nrow(df)

# number of columns in your selected dataset

numcols<- ncol(df)

# z will store the outputs

z<-(1:numrows)

# q is a vector for representing the  group of array

q<-(1:numcols)

# for example, if first 6 samples are from same group(normal) and
# next 6 are from another group(tumor)

q<-c(1,1,1,1,1,1,2,2,2,2,2,2)

for(i in 1:numrows){
s1<-c()
s2<-c()
k1<-c(df[i,])

k2<-as.numeric(k1)
for(j in 1:numcols){
if(q[j]==1) s1<-c(s1,k2[j])
if(q[j]==2) s2<-c(s2,k2[j])
}
s1<-as.numeric(s1)
s2<-as.numeric(s2)

z[i]<-t.test(s1,s2)$p.value
}

#add result column as data.frame
#append a column to the active dataset or create a new subset of data

addResultColumns(data.frame(z),c("t-test_P-value"))
#################################################################
```

Script 5:

```
#################################################################
# Estimation of p-values for each gene, from a set of
# permutations in a SAM analysis

# data.frame for storing selected columns

df<-getDataset()

# y is a vector for representing the class of array
# for example, if first 6 arrays are from same class(normal) and
# next 6 are of another class(tumor)

y<-c(rep(1,6),rep(2,6))

# load the package samr for SAM analysis

library(samr)
data=list(x=df,y=y,logged2=TRUE)

# samr(data,  resp.type=c("Quantitative","Two class unpaired",
# "Survival", "Multiclass",
# "One class", "Two class paired", "Two class unpaired time course",
# "One class timecourse", "Two class paired timecourse",
# "Pattern discovery"),
# s0=NULL,
# s0.perc=NULL, nperms=100,
# center.arrays=FALSE, testStatistic=c("standard", "wilcoxon"),
# time.summary.type=c("slope", "signed.area"),
# regression.method=c("standard", "ranks"),
# return.x=FALSE, knn.neighbors=10, random.seed=NULL,
# xl.mode=c("regular", "firsttime", "next20", "lasttime"),
# xl.time=NULL,  xl.prevfit=NULL)

# data: Data object with components x- p by n matrix of features,
# one observation per column (missing values allowed);
# y- n-vector of outcome measurements;
# censoring.status- n-vector of censoring censoring.status
# (1= died or event occurred,0=survived, or event was censored),
# needed for a censored survival outcome

# resp.type: Problem type: "quantitative" for a continuous parameter;
# "Two class unpaired" ; "Survival" for censored survival outcome;
# "Multiclass" : more than 2 groups;  "One class" for a single group;
# "Two class paired" for two classes withpaired observations;
# "Two class unpaired timecourse", "One class time course",
# "Two class.paired timecourse" or "Pattern discovery"

# s0: Exchangeability factor for denominator of test statistic;
# Default is automatic choice

# s0.perc: Percentile of standard  deviation values to use for s0;
# default is automatic choice; -1 means s0=0
# (different from s0.perc=0, meaning s0=zeroeth percentile of
# standard deviation values= min of sd values

# nperms: Number of permutations used to estimate false discovery rates
# center.arrays: Should the data for each sample (array)
# be median centered at the outset? Default =FALSE
```

```
# testStatistic: Test statistic to use in two class unpaired case.
# Either "standard" (t-statistic) or ,
# "wilcoxon" (Two-sample wilcoxon or Mann-Whitney test)

# time.summary.type: Summary measure for each time course:
# "slope", or "signed.area"),

# regression.method: Regression method for quantitative case:
# "standard",(linear least squares) or
# "ranks" (linear least squares  on ranked data)

# return.x: Should the matrix of feature values be returned?
# Only useful for time course data, where x contains summaries
# of the features over time. Otherwise x is the same as the
# input data data$x

# knn.neighbors: Number of nearest neighbors to use for
# imputation of missing features values

# random.seed: Optional initial seed for random number generator (integer)

# xl.mode: Used by Excel interface

# xl.time: Used by Excel interface

# xl.prevfit: Used by Excel interface

# performing the Man-Whitney test

samr.obj<- samr(data, resp.type="Two class unpaired",testStatistic="wilcoxon", nperms=100)

# compute the p-values of each gene
# pv=samr.pvalues.from.perms(samr.obj$tt,samr.obj$ttstar)

# Computes tables of thresholds, cutpoints and corresponding False Discovery rates for SAM (Significance analysis of
microarrays) analysis.

# samr.obj: Object returned from call to samr
# min.foldchange: The minimum fold change desired; should be >1; default is zero, meaning no fold change criterion is
  applied
# dels: vector of delta values used. By default, 50 values are chosen in the relevant operating change for delta. Delta is
  vertical, the distance from the 45 degree line to the upper and lower parallel lines that define the SAM threshold rule.
# nvals Number of delta values used.

d = samr.compute.delta.table(samr.obj, min.foldchange = 1.2, dels=NULL, nvals = 50)

# add result column as data.frame
# append a column to the active dataset or create a new subset of data

addResultColumns(data.frame(d),c("SAM-value"))

################################################################
```

Conclusion

Using the methods described in this Technical Note users can integrate various R-scripts and functions into the ArrayAssist software, thereby extending the analysis and visualization capabilities beyond those specified in the graphic user interface.

For more information on this Technical Note
or to contact our Technical Service department:

Stratagene US and Canada
Technical Service: 800-894-1304 x2

Stratagene Europe
Technical Service: 00800-7400-7400

Stratagene Japan K.K.
Technical Service: 3-5821-8076

Visit our web page at:
www.stratagene.com/contacts/TechServices

Email our Technical Service department at:
techservices@stratagene.com

www.stratagene.com