# Rlabkey Users Guide

| | |
|---|---|
| Rlabkey version 2.1.xxx<br>LabKey Server version 10.1 | Author: Peter Hussey<br>peter@labkey.com<br>Document first release March 28, 2010 |
| This User Guide supplement s the help files on the individual functions of Rlabkey. It includes an overview of the integration between R and LabKey Server. It also covers specific topics that require interaction with the LabKey Server via the browser. Note that the screen shots of LabKey Server user pages may be different on the version of LabKey to which you are connecting. | |

## Working with R and LabKey Server together

Rlabkey is an interface between the R language and LabKey Server that has been designed to combine the strengths of LabKey Server and the R language platform. R has emerged as a leading open source analysis tool for bioinformatics. The combination of the R language, its packaging and distribution architecture, and an active international community of contributors have created a body of statistical and bioinformatics functionality that is widely used and constantly improving. LabKey Server is also an open-source tool for bioinformatics that helps researchers organize, analyze, and share their research data. LabKey Server is designed to manage the huge range of sizes and formats of data files generated by research instruments. LabKey Server also makes its data accessible to popular analysis tools, including R, SAS, and Excel. By facilitating both input and analysis, LabKey Server acts as a data hub for groups of researchers, whether they are in the same lab or collaborating across continents.

R was first integrated into LabKey Server as a reporting and visualization tool. "R Views" are scripts run by LabKey Server. They are defined in the context of a specific grid view of data through the View..Create menu in the data grid. In an R View, the current dataset shown in the grid is available to the script implicitly as a data frame named labkey.data. R Views are commonly used to produce graphs or summary reports that are incorporated into web pages. LabKey then released the first version of the Rlabkey package (version "0.0") that was designed to enable R scripts running on a client machine to read and write data in LabKey. The second major release of the Rlabkey package (renamed version "2.1") added support for the interactive R user . The new functions in the Rlabkey package enable R users to explore and analyze the data stored in LabKey entirely from the R command prompt. Researchers and statisticians can also use Rlabkey to save their results back to LabKey Server, enabling tracking and aggregation of results over multiple runs. The combination of R Views and the Rlabkey package makes R useful in many roles on a LabKey Server, as depicted in Figure 1.

For more information, see The R project for Statistical Computing (www.r-project.org) and LabKey Software (www.labkey.com) .

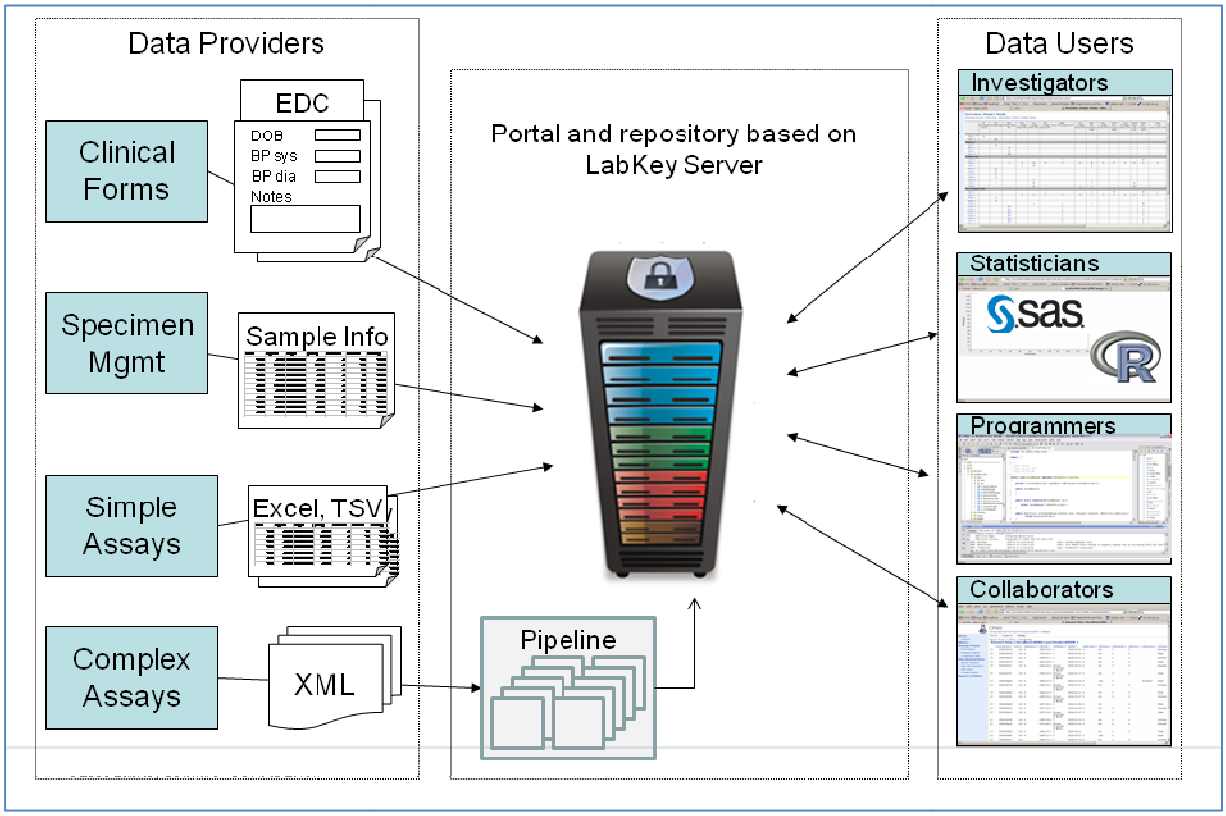


**FIGURE 1.**

## Overview of the Rlabkey API

There are two basic types of functions available in Rlabkey. The "stateless" functions have names that begin with the prefix "labkey." The parameters of these stateless functions contain all the information required to read or update their target data. The labkey.SelectRows function is a good example:

```
rows <- labkey.selectRows(baseUrl="http://localhost:8080/labkey",
                          folderPath="/apisamples",
                          schemaName="lists",
                          queryName="AllTypes")
```

The parameters to labkey.SelectRows specify the server address, the project and folders within the server that are the context of the rows to select, plus the schema name and query name of the data to retrieve. Additional parameters allow specification of column lists, row filters and sort order of the returned data frame. Almost all of the parameters to the stateless functions are specified as strings.

In contrast, the session-based functions in Rlabkey version 2.1 are designed to be used in sets. A simple example that retrieves the same result as the function above is the following:

```
session<-getSession(baseUrl="http://localhost:8080/labkey",
                    folderPath="/apisamples")
schema <- getSchema(session, "lists")
rows<- getRows(session, schema$AllTypes)
```

There are two primary advantages to the session-style functions for an interactive R user:

- **Efficiency:** The user need only specify the baseUrl and folderPath once, and it is saved in the session object where it can be used to do multiple selects or updates against one or more data sets. The function names are shorter to save typing. Also, the arguments to the session-based are usually object-valued, allowing the R editor to provide statement completion choices as the user types.
- **Discovery:** The object returned by getSchema is a list of *all available* query objects within that schema on the server. These query objects in turn are a list of their fields and field attributes. These schema objects allow the R user to discover what is available on the server. Field attributes for "lookup" fields connect to other queries, effectively allowing the user to specify joins without knowing SQL.

All of the Rlabkey APIs can be used in an R View script as well as on client machines.

## User logins and passwords

To access most data on LabKey Server, the user must be logged in to LabKey and have the appropriate permissions to the data. (The exception is when the anonymous "Guest" user has been given access permission. See security topics for details.) Rlabkey connects using login information stored in a netrc file. The netrc file contains configuration and login information for the File Transfer Protocol client (ftp) and other programs such as CURL.

On a UNIX system this file should be named .netrc (dot netrc) and on windows it should be named _netrc (underscore netrc). The file should be located in the user's home directory and the permissions on the file should be unreadable for everybody except the owner.

To create the _netrc on a windows machine, first create an environment variable called 'HOME' that is set to your home directory (c:/Users/<User-Name> on Vista) or any directory you want to use. In that directory, create a text file named _netrc (note that it is underscore netrc, not dot netrc like it is on UNIX).

The following three lines must be included in the .netrc or _netrc file either separated by white space (spaces, tabs, or newlines) or commas.

```
machine <remote-machine-name>
login <user-email>
password <user-password>
```

One example would be:

```
machine localhost
login peter@labkey.com
password mypass
```

Another example would be:

```
machine atlas.scharp.org login vobencha@fhcrc.org password mypass
```

Multiple such blocks can exist in one file. Also , a netrc file is not used by functions in an R View script.

## Using Rlabkey to save analysis results

You must define an assay on the LabKey Server in order to use the saveResults function. The following steps go through the basic steps. (See the Assays topic on LabKey.org for more details.)

1. In R, write out a sample result table as a tab-separated text file without NA indicators, quotes or row names

```
write.table(topTable, "c:/temp/limmaResults.txt",
            sep="\t", na="", quote=FALSE, row.names=FALSE)
```

2. Login to LabKey Server via a browser
3. If you don't already have a project to work in, define a new project where the results will be stored. Choose the Folder type of "Assay ".

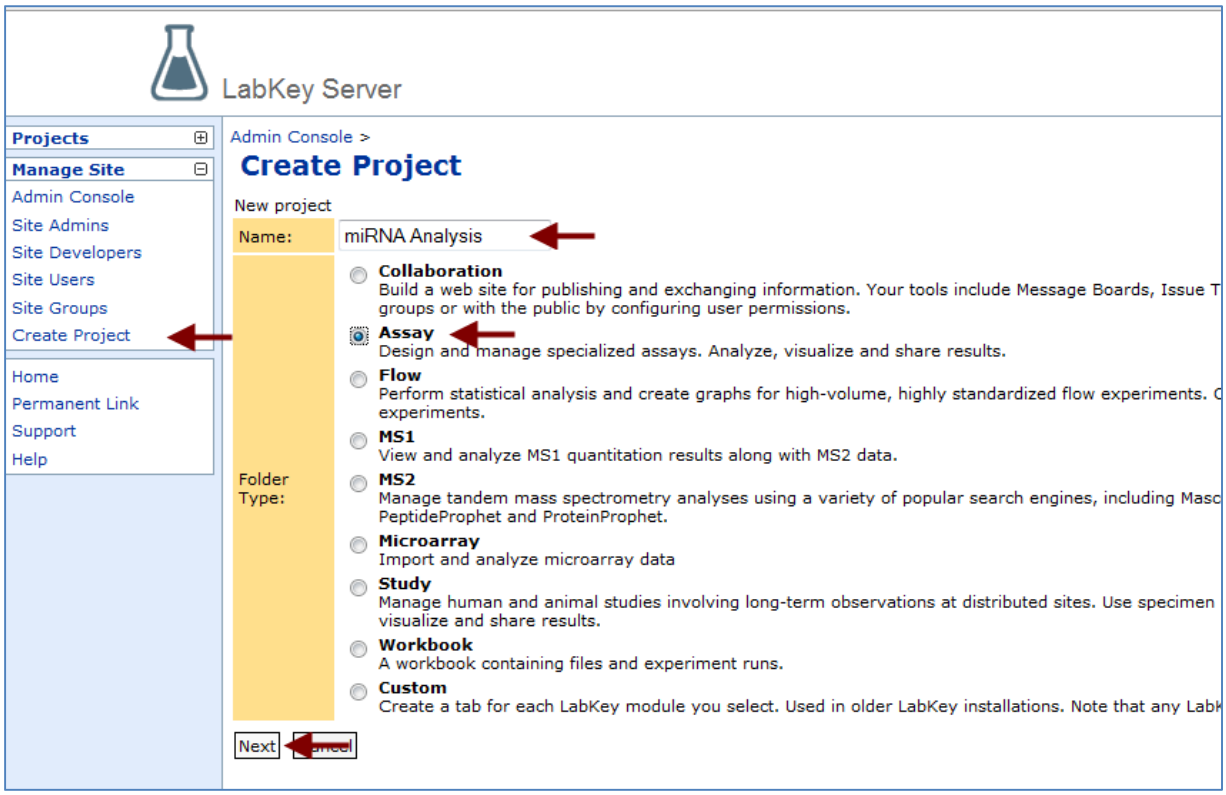


**FIGURE 2**

Leave the permission settings at their defaults for now.

4. On the home page (also called the "Dashboard") of the project in which the assay results will be stored, you should have an Assay List. (If one is not visible, use the drop-down adjacent to the Add

Web Part button to select Assay List and then click the button.) On this Assay List, select the New Assay Design button.

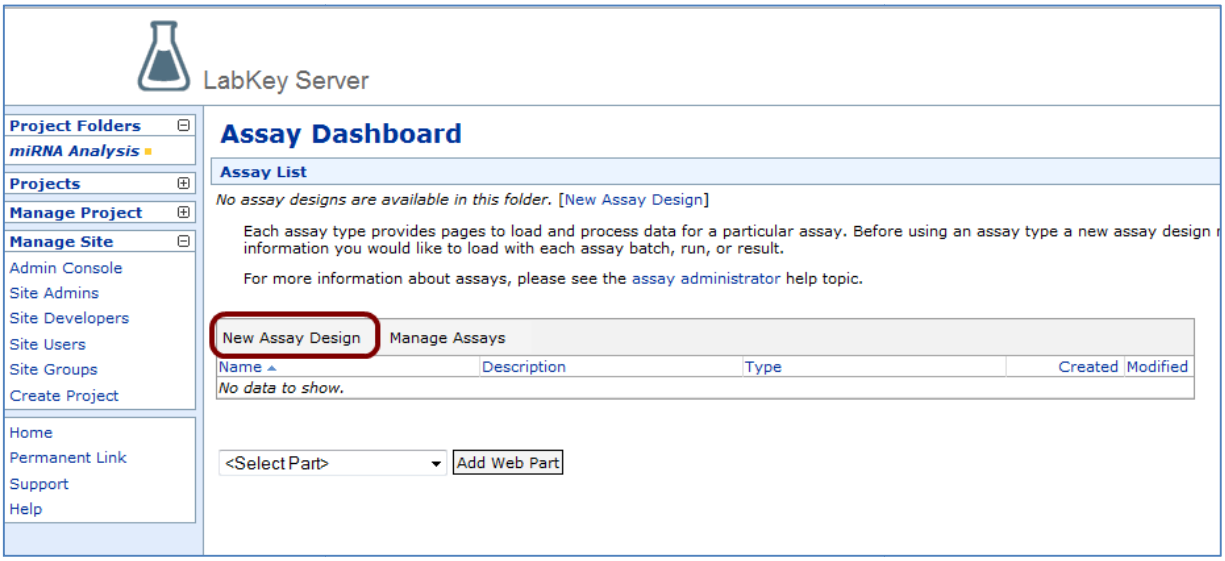


**FIGURE 3**

Then select the "General" assay type from the list and press Next:

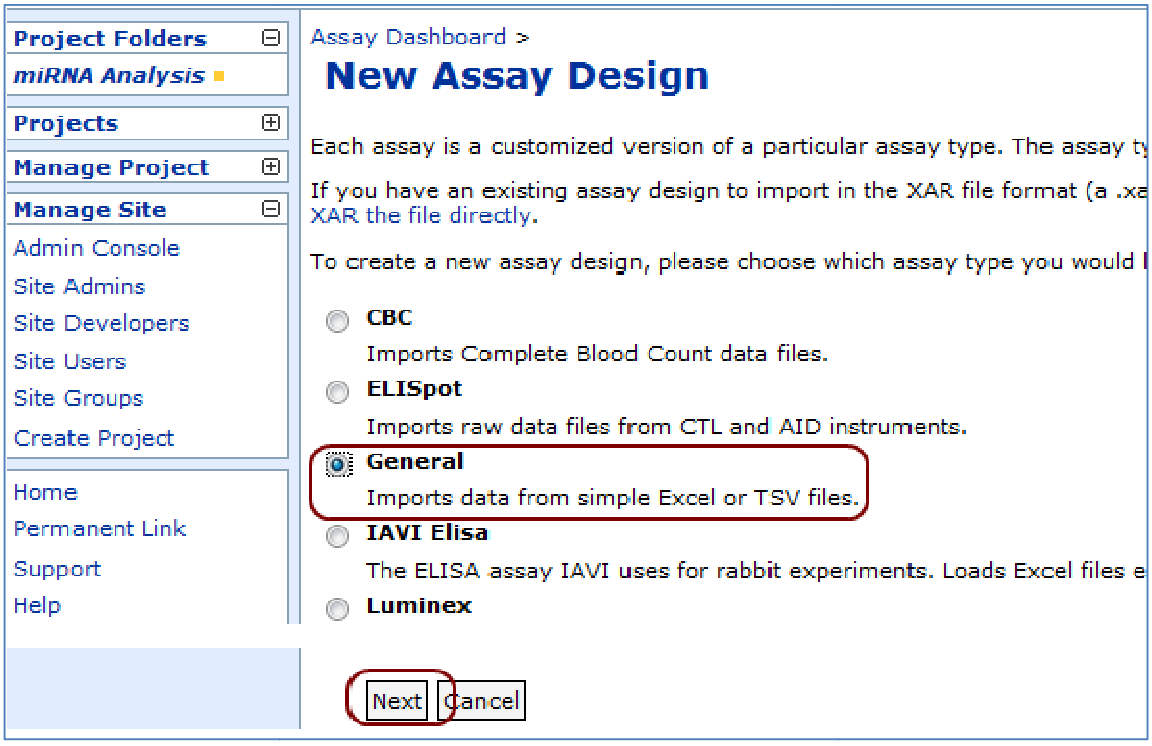


**FIGURE 4**

5. You now are presented with the General Assay Designer page.

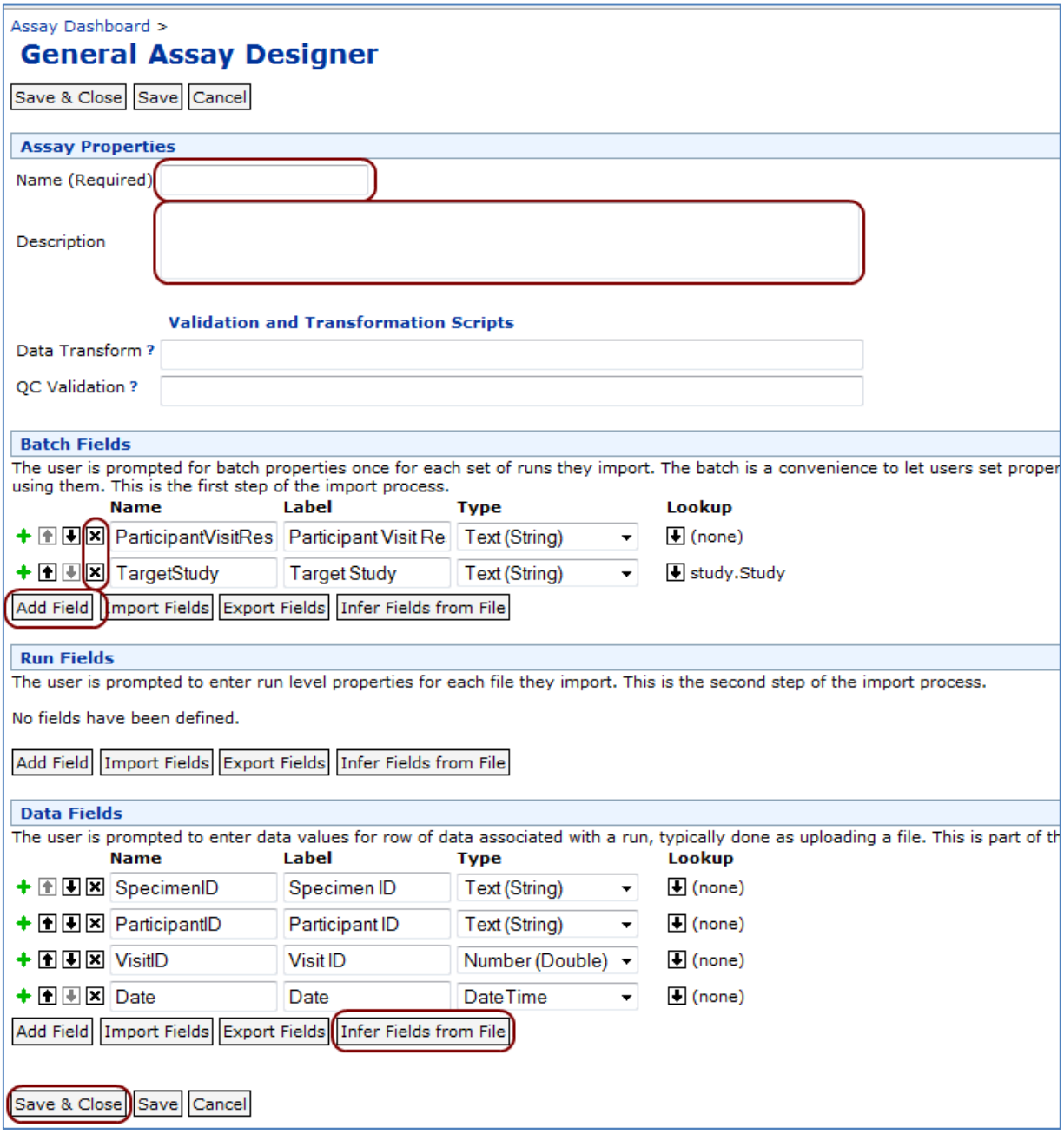


Figure 5

This page is broken into four sections, namely the properties of the assay definition itself, metadata properties that apply to a "batch" of one or more runs, metadata properties of individual runs, and then the data properties that are the rows of the data frames (result sets) to be uploaded. "Metadata" refers to property values describing the results as a whole, including any necessary information about how they were calculated that is not recorded in the data frame itself.

In our example we will be assuming a single run per batch, and we will record two example properties recording the versions of the R packages used, just for illustration.

a. Give the assay a name and a description. Avoid spaces or special characters in the name.
b. In the Batch Fields section,
    i. Remove the pre-defined Batch properties by clicking the X next to each
    ii. Press Add field to add text fields named LimmaVersion and RlabkeyVersion.

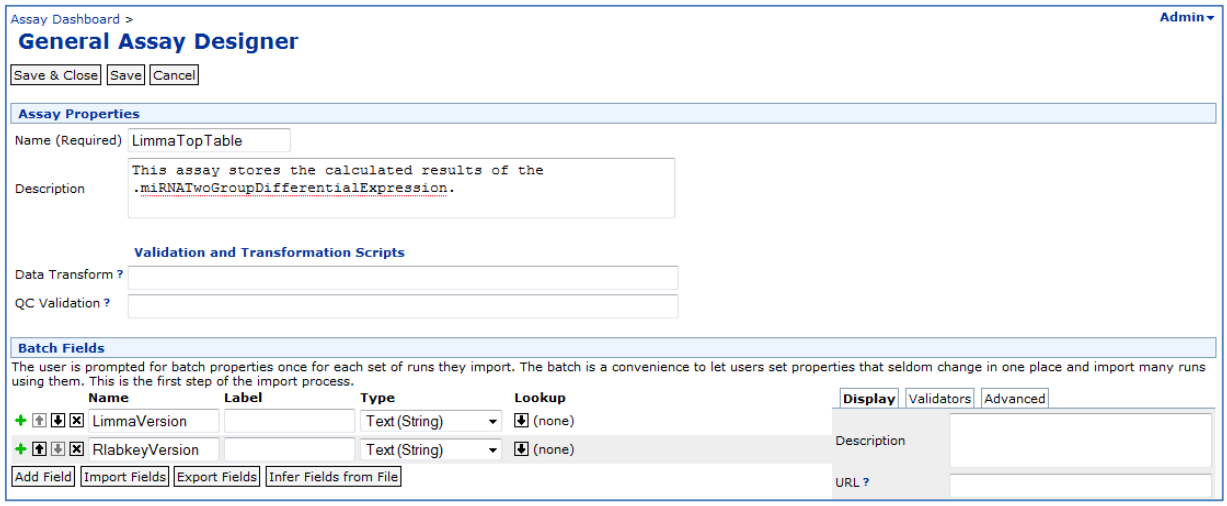

Figure 6

c. In the Data Fields section,
    i. Press the Infer Fields from File button (OK to prompt of deleting existing fields ).
    ii. In the Infer Fields dialog press the Browse button and navigate to the file written out in step 1. Then press Submit. The bottom part of the page should now look like

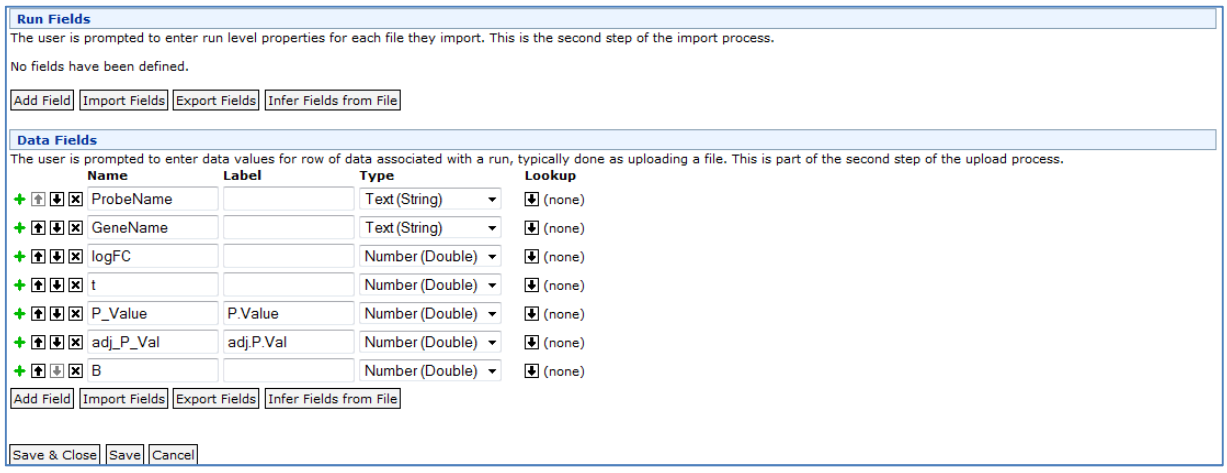

Figure 7

If the column names of your data frame contained characters that aren't allowed in the Data Field names of an assay, they should be replaced by underscores as shown above. If the field names and

types inferred by LabKey Server look correct, press Save & Close. Your newly defined assay should now appear in the list:

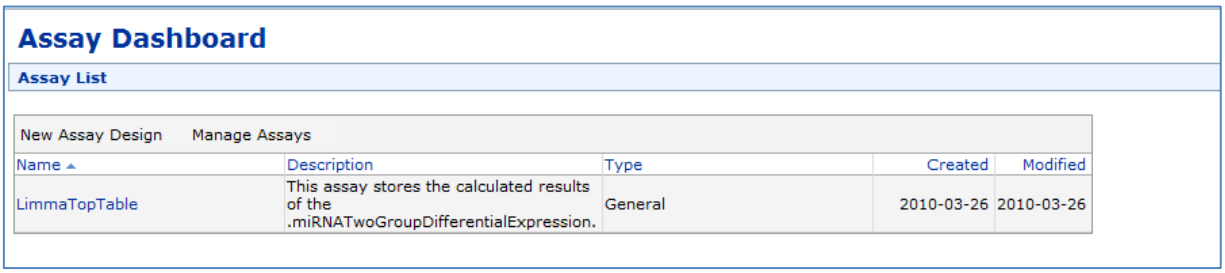


Figure 8

6. You are now ready to save results sets from R. The highlighted names in the code block below correspond to the names you defined in the preceding steps. Note that there is a built-in property of Batch Name that you will likely want to define. In this code the batch name will be "Batch <timestamp>". Also note that the topTable rows being saved are only those which have a B value greater than 0.

```
s<-getSession(baseUrl="http://localhost:8080/labkey", folderPath="/miRNA Analysis")

## get the versions of the currently loaded limma and Rlabkey libraries,
## taken from installed.packages()
>ipdf <- data.frame(installed.packages(), stringsAsFactors=FALSE)
>bprops <- list(LimmaVersion=ipdf["limma", "Version"],
           RlabkeyVersion=ipdf["Rlabkey", "Version"])

## format as a list of lists
bpl<- list(name=paste("Batch ", as.character(date())),properties = bprops)

## call saveResults
>assaybatch <-saveResults(s, assayName="LimmaTopTable",
            resultDataFrame=topTable[topTable$B>0, ],batchPropertyList= bpl)
```

After saving two assay results in this way, clicking on the name of your assay in the Assay List will bring up the Runs view listing the datasets you have saved. Click on the Assay Id link to see the results of a single run, or the "view results" link to see the data across multiple runs. These data can also be queried and retrieved from R. In this example, they will belong to a data set named "LimmaTopTable Data"

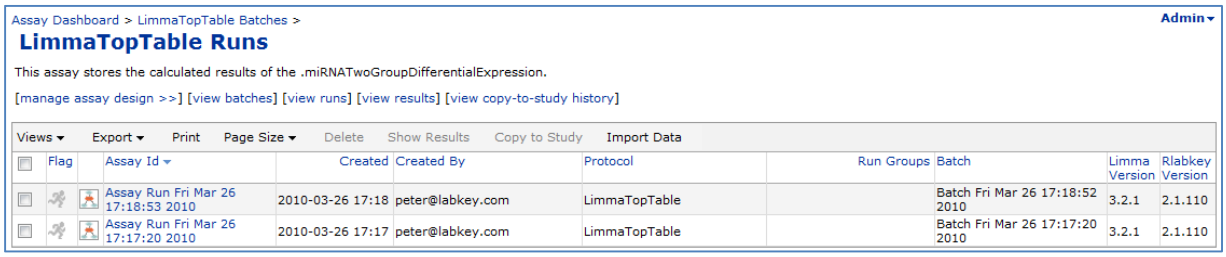


**FIGURE 9**

## Organizing multiple assay results

You may want to create separate folders within a project to hold results for different users. Segregating results into user-specific folders can be useful for security restrictions (not all users are allowed to view other user's data), to allow QC review before consolidation, or for simple organizational reasons. For consistency across subfolders, all users should use the same Assay, defined at the project level as described above. To refer to this definition, users must be given read permissions at the project level.

To create subfolders, use the Folders link under Manage Project

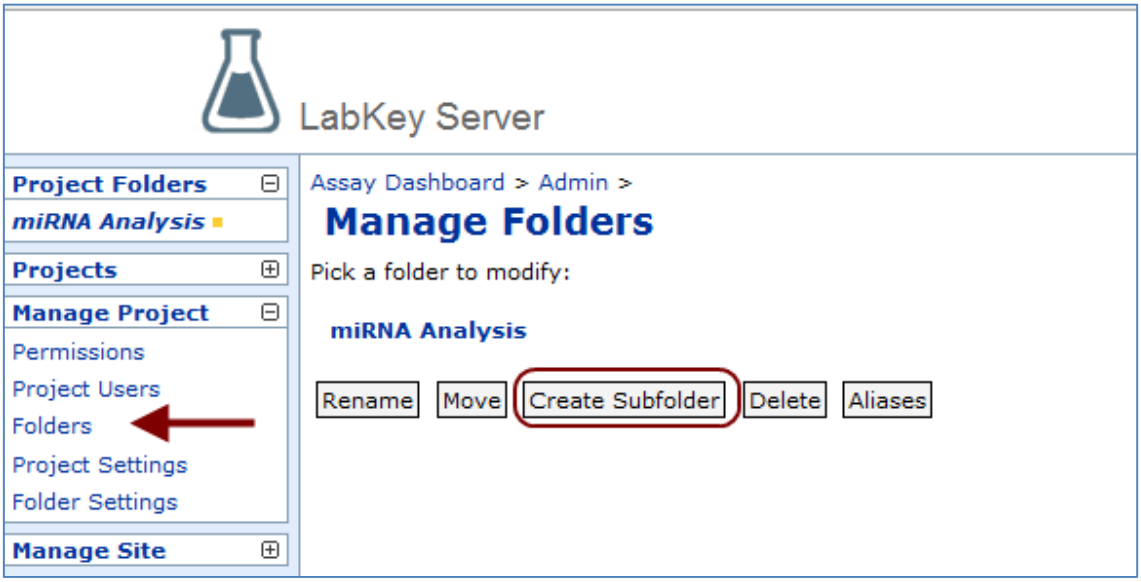


Figure 10

To save results to a folder named "johnd" within the project described in the previous section, the only difference in the R code is to use the full path to the subfolder:

```
s<-getSession(baseUrl="http://localhost:8080/labkey",
                              folderPath="/miRNA Analysis/johnd")
```