

### \*\*\*\* Mev+R Programming API Introduction \*\*\*\*

The main steps for creating an MeV+R module are as follows:

1. Format the user's data using RDataFormatter
2. Get an Rserve Connection from RconnectionManager
3. Load user's data into R
4. Execute any commands necessary for your particular R package
5. Retrieve resultant data
6. Display results through MeV Viewer

The preceding six steps are accomplished through three common software elements. Each implementation will require: a data structure translation from MeV's model to R's model (step 1); a communication layer between MeV and R (steps 2-5); and visualization outputs in MeV (step 6). Our data translation wrapper, connection classes, and some visualizations can be used virtually unchanged. New module developers will need only to develop specialized data visualizations for complex or non-standard data views using MeV's built in API.

We've created an MeV data structure wrapper that formats for R. It is flexible for use with either single or dual color data. It is easily instantiated, taking only MeV's data structure as a parameter. There are also a few convenience classes (optional) available to the developer that aid in keeping track of the user's data in its various states of dye type and orientation.

The R to MeV connection is built and can be called with no modifications. It is written in the form of a singleton. The developer instantiates an Rconnection object and simply calls getConnection(), passing only an IP address and port # for the R server. The object returns an existing connection if one exists, creates one and returns it if none exist, or pops up a dialog giving the user instructions and options on how to proceed. It should also be noted that we provide some additional code (optional) that can be included in a module's dialog box, allowing the user to save IP and Port number information automatically in their MeV config file.

Finally, developers will need to display the results of the analysis back to the user as MeV *Viewers*. There are a handful of basic visualizations pertaining to microarrays that either built into the MeV API, or we have built for simple adaptation. Most often, the developer will simply extend off a *Viewer*, passing in a matrix of intensity or ratio values and an annotation matrix if one exists. More complex visualizations are also easily implemented using MeV's built in *Viewer* API.

#### \*\*\*\* Overview \*\*\*\*

The MeV-R integration is composed of 3 parts: MeV, R, and Rserve. MeV contains a package (org.tigr.microarray.mev.r) that contains a set of java classes that can talk with R. R is installed on the user's machine. Rserve provides the communication link between MeV (Java) and R. It is a TCP/IP server. By default, it runs on port 6311, but this can be changed in a config file.

#### \*\*\*\* Integration Details \*\*\*\*

Once R and Rserve are installed, a developer can work purely in Java. You simply pass the exact commands that you would issue at a command-line in R through the Rconnection. Retrieving results is a simple matter of declaring a type and issuing a command. Of course, the types are not exactly matched, so the developer may need to do some converting from a double to a float, etc.

#### \*\*\*\* Code You Could Use \*\*\*\*

In package : org.tigr.microarray.mev.r

I've written a couple of reusable classes for MeV and R:

1. RconnectionManager - A singleton class for managing connections
2. RDataFormatter - Formats an instance of MeV's IData interface for loading into R. This class basically reads the data matrix that a user loaded into MeV into a single long, comma-delimited string that can be passed into R. Once in R, the array of data must be re-dimensioned into the original matrix. The former is accomplished by a single statement  

```
'dim(arrayName) <- c( #genes, #arrays)'
```
3. RHyb - A convenience class that holds all the important information about each of the user's loaded arrays (slides or hybs). The tricky thing with this class is that it is generalized to deal with both 2-color or intensity data. If it is 2-color, it wants to know what color dye was used to label what sample (treated or control). If it is intensity, it wants to know if the hyb in question is control or treated.
4. RHybSet - A set of RHybs
5. \*ClassAssigner - This may be useful if you need a GUI for the user to assign something to each and every loaded array. We use it in our classification algorithms so the user can tell us what class each array belongs to if any. It also allows the user to save their selections to a file which can save a lot of time in the long run (especially for testing!)
6. GenericFileDialog - pretty self explanatory

#### **\*\*\*\* Development Process Pseudocode \*\*\*\***

So the coding process goes something like this:

1. Format the user's data using RDataFormatter
2. Get an Rserve Connection from RconnectionManager
3. Tell R what libraries to load
4. Load user's data into R
5. Re-dimension the data
6. Execute any commands necessary for your particular R package
7. Retrieve resultant data
8. Display to user through MeV Viewer

Obviously, the bulk of the coding is in developing the Java GUI and wrestling with Swing. I have done some re-usable work in this area with Viewers and File saving.

#### **\*\*\*\* Thread and Access Issues \*\*\*\***

The tricky part of dealing with this integration is in threading and error trapping.

You don't have access to R's output unless explicitly called. Therefore, you can't do things like update a determinate status bar or pass along error messages. The best I've been able to muster is to estimate the length of processing time based on the number of cycles each process takes.

In terms of threading, the R process doesn't exactly kick off a message telling you that it's done, so I kick off the R calls in a separate thread with a wait loop monitoring the R process.

#### **\*\*\*\* Important Rserve information \*\*\*\***

You will want to pay some attention to version issues between R and Rserve.

They also have a development version of Rserve that gives more verbose messages.

#### **\*\*\*\* Windows Issues \*\*\*\***

Rserve is not perfect on Windows! There are some namespace issues with the server which prevents Rserve from supporting multiple connections. This is not a huge issue as we have developed this integration with single users installing everything on their own machine in mind. If you are considering hosting an Rserve server it should be done on a \*nix machine. See the Rserve website <http://stats.math.uni-augsburg.de/Rserve/> for details