# Documention for Editor

Thomas Zamojski

November 6, 2015

This is an implementation of an Editor in a client/server architecture. Clients can connect to the server and launch an editor, which state is kept on the server. All clients registered on the server have access to the same ressources and share a unique state on the server. Moreover, the editor has the capability to evaluate R code and output the result locally for clients. That is, the output is not sent to all clients, but only to the client requesting the evaluation. This choice was made solely to show how to use REST to implement such behaviour over RMI, which is already shown with the state notifications.

The project consists of two parts: the server and the client. The communication between client/server is done through a unique interface, and in two ways:

- RMI for (un)registering clients on the server, and for the server to send notifications of state changes to all registered clients.
- restful architecture for http requests from clients to the server for all other state changing requests.

## 1 Interface

For the rest architecture, the interface is the following: the server makes all resources available at "host":8080/engine/, where "host" has to be specified (localhost for e.g.). Resources are:

| | |
|---|---|
| /cut | Get request to cut the selection. Return nothing. |
| /copy | Get request to copy the selection. Return nothing. |
| /paste | Get request to paste the clipboard. Return nothing. |
| /setSelection | Post request with a JSON of an int array with the selection start and length arguments that the server records. Return nothing. |
| /insert | Post request with the plain text to be inserted in the buffer on the server. Return nothing. |
| /contents | Get request. The server returns the content of the editor buffer as plain text. |
| /evaluate | Get request. The server parses and run R code of the buffer and returns the output as plain text. |

/init Get request for the client to notify the server that it is conneting.

For the RMI interface, it consists of two java interfaces that must be shared among all server and clients:

- Observer: client side interface.
- EditorEngine: server side interface.

## 2 Server's Implementation

Launch: using RestServer.java, with security set in the security.txt file. The services provided by the server are included in two classes:

- EngineSingleton: The data and the methods to operate on the data.
- EditorEngineImpl: Communication between the client and EngineSingleton.

The EngineSingleton follows the singleton pattern: only one instance of the class is instanciated. This is to assure that all clients share the same data for the editor. When the singleton is instanciated, it also registers itself into the RMI register and initiate a renjin engine. The data stored on the server is the following:

- StringBuffer for the contents of the editor.
- Two integers for the selection start and selection length, specifying a selection.
- String for the clipboard (temporary buffer for cut/copy/paste).
- Renjin engine that performs the interpretation of R code.
- List of Observers for registered clients.

Furthermore, the singleton also implements the methods to manipulate the data: cut, copy, paste, insert, contents, setSelection and evaluate. Moreover, it extends EditorEngine, so has also addObservers and removeObservers methods. Adding observers is simply adding to the list. Removing observers is implemented as removing from the list, but not only. If no more observers are connected to the server, it shuts down the singleton, so that data is not carried through future sessions.

The Rest interface is implemented in EditorEngineImpl. Its purpose is to accept http requests from the clients, redirect them to the singleton and response to the client.

The server is initialised by launching RestServer.java. Then, it follows two steps:

- First it starts the rest server making EditorEngineImpl's services available for http requests on port 8080. It also creates a RMI registry, accessible on port 9999.

- Whenever a first client connects to the server, it instanciate an EngineSingleton singleton and registers it in the RMI registry under the name "engine". It also launches the renjin engine.

This second stage is undone whenever there is no more clients, so that the data is not persisted automatically.

# 3 Client's Implementation

Launch with SimpleEditor.java, security are set in security.txt.

The client has two parts:

- GUI: graphical user interface including menus for actions, a text buffer in the upper part and an output buffer on the lower part.
- Controller: Controls the communication between the user's actions in the GUI and the server.

IMPORTANT: For the current implementation to behave correctly, to quit the application, use Menu–File–Exit and not the x button in the upper left corner.

The GUI has a listener on the keyboard and mouse, so the user can use the editor as with usual editors. Unfortunately, the current version does not implement backspace, enter and other special keys. (By the way, javaFX seems to have a strange treatement of Typed events.) Selection shown in blue is not synchronized with other clients. It is recorded on the client side, and sent to the server only when using it in an action (e.g. copy).

One can use evaluate to interprete R code whose output is then printed in the output buffer. The code evaluated is the selection or all the input buffer if there is no selection.

On initiation, the controller sends an init request to the server. This assures the server to be initiated and registered in the RMI. Thus it is assumed that the rest server is already up, but not the RMI. It then looks for the server in the RMI registry and adds itself as an Observer in the server.

Then the controller listens for actions on the GUI from the user, and builds the corresponding http requests according to the rest interface, except for exit, which does a removeObserver using RMI.

Finally, one launch a client using SimpleEditor.java, and quits by clicking on Menu–File–Exit.

# 4 Thoughts about future versions

The selection notifications have been disabled on the server. This prevents having a huge number of notifications and http requests being created whenever one types characters or makes a selection using the mouse. Another solution would be to implement a less active communication procedure between clients and servers in that case.

Also one could implement saving files to the filesystem, as well as some history of commands sent to renjin.

The use of javaFX for typing is unusual to the author, and couldn't implement here special characters such as return or backspace, or ctrl-v and so on.