RAD Documentation

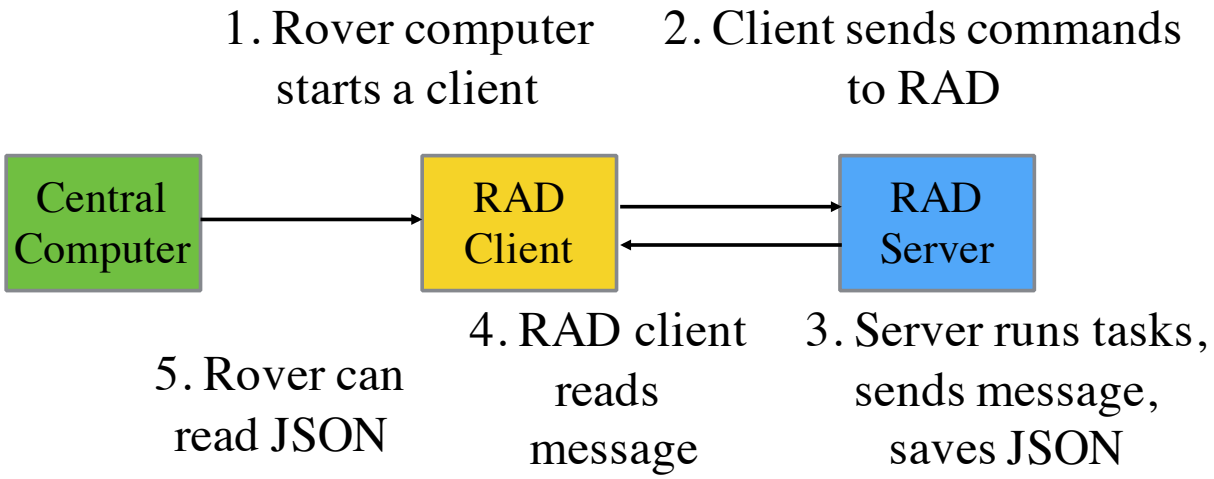
To start the RAD, make sure that the JSON Simple and GSON JARs are available in the project structure.

The real RAD changes from sleep to science mode and vice versa. It sleeps for 45 minutes and wakes for 15 minutes to record radiation data. It then sends the data once a sol (day) to the main computer. To store data we are using the JSON file as a sort of cache. It then adds entries each scan.

Commands

```
RAD_OFF      This will stop all power from going to the RAD

RAD_BOOTUP   Boots up the system (fully turns power on)

RAD_SCIENCE  Science data is collected for 15 mins

RAD_CHECKOUT Send data to rover, once per sol. (However, this also
    removes the JSON data stored in the file system to develop new
    data for each day. It's best to read the JSON first, then call
    this method.)


RAD_SHUTDOWN Loads sleep timer and initiates sleep state

RAD_SLEEP    In sleep mode it waits for sleep duration to complete (45
    mins and then goes to BOOTUP

RAD_IS_ON      Asks the RAD if it is on?

RAD_GET_POWER  Get RAD power consumption (However, this is stored in
    the JSON file, and can be read from there.)
```

Workflow



Here we see the central computer starting a client to communicate with the server. The server receives messages from the client, and stores data in JSON format. The computer can then read this data. Meanwhile, the client receives back a message from the server as a form of acknowledgement.

Java Classes

Rad.java

Contains the main component. It has constants, states, and a collection of data. It also has a few methods for querying the state of the RAD. Data is stored as a `HashMap<Long, Double>` that represent a date in milliseconds and the radiation level recorded at that time.

RadMain.java

Starts the main components. Creates server and client instances with the same port.

RadClient.java

The client sends the server messages. The `sendMessage` method is used to send messages to the RAD server.

RadServer.java

The server receives messages and does tasks according to the incoming messages. The `doWork` method determines the Rad's current state. When the RAD is sleeping, it sets a timer to go back to science mode after 15 seconds (not the real 45 minutes).

When the RAD is in science mode, it scans for radiation data. The data is then saved into a JSON file.

To keep aggregating data, the server first reads the current JSON file. It parses the data, and sets it to the current Rad instance.

Example

The RadClient is actually an example of how to communicate to the RAD server. It can represent the main computer or the battery component, which can turn off the RAD to preserve energy.

RadClient.java

Within the `run` method we have several commands to send to the server.

```
sendMessage("RAD_BOOTUP");

sendMessage("RAD_IS_ON");

sendMessage("RAD_GET_POWER");

sendMessage("RAD_CHECKOUT"); // get data as a message and clear it.

sendMessage("RAD_SHUTDOWN");

sendMessage("RAD_OFF");

sendMessage("RAD_IS_ON");

sendMessage("exit");
```

These messages tell the RAD server to 1) start the RAD, 2) ask if the RAD is on, 3) ask how much power it is using, 4) checkout the data (however, this actually just deletes JSON data, so it's best to read it first, before doing this, as it should only be done once a sol), 5) shut down the RAD, which makes it go into sleep mode, 6) turn off the RAD to remove all of its energy, 7) ask again if the RAD is on, and 8) turn off the server.

These messages can be set up to be used by the main computer or the battery.

RadServer.java

To duplicate real values, we use the following code to get 15 values in the doWork method. These values are then added to the data. They are within our constraints of .1 to 10^4.

```
for (int i = 0; i < 15; i++) {

                        double calc = Rad.MIN_RADIATION
                                    + (Math.random() * ((Rad.MAX_RADIATION
- Rad.MIN_RADIATION) + 1));

                        rad.addMeasurement(calc);
                }
```

Running the application

To run the application start the RadMain.java file in Eclipse. It should then have the client send the server the messages.

It should connect the server and client, and then start the communication between them.