

JBora Quick Guide

Packaging:

The `jbora` package contains the JBora classes used by applications. This package is linked to the Spread which is the underlying group communication tool. The two examples are contained in a separate package called `testjbora`.

Installation:

First, you have to install and configure Spread on each server machine. Next, make sure that the packages `jbora`, and `spread` are in your classpath (the latter is the Java interface contained in the Spread distribution). "Our" packages can be found in the directory *JBora_home/classes*. The Java interface to Spread is located in *Spread_home/java/*. To make this part simpler, we provide two jar files in *JBora_home/lib*: *jbora.jar* containing the JBora package and *spread.jar* containing the Java interface to Spread. You may download Spread from: www.spread.org.

Overview:

The two simple examples briefly illustrate the main features of the JBora middleware: sending and receiving multicast messages; receiving view changes; inter-thread communication; whiteboard usage and propagation.

Running the first example:

The source code for the classes involved in both this and the next example can be found in *JBora_home/src/testjbora*. The key lines of the code are emphasized below, but please take a few minutes to look at the source files as well. The first example works with the following classes: *Daemon.java*, *Test.java*, *WBHandler.java*, *IntMessage.java* and *StringMessage.java*. The last two classes are just containers for sending Integer and String values as JBora messages.

This example consists of a set of identical processes that we call “daemons” (not to be confused with the Spread daemon). Each such process simply prints on its standard output the view changes and messages received but it does not multicast any message.

To run this example, first, start the Spread daemon on each host that will participate to the test. Then, from your *JBora_home/classes*, you may start our daemon as follows:

```
java testjbora.Daemon
```

You may start more than one daemon per host. Connection name for each host is *connXXXX* where *XXXX* is a four digit random number. Each daemon (group member in general) is identified by the JBora middleware as *conn_name@host_name*.

Each daemon receives all messages sent to the group (multicasts) as well as the membership information (view changes) sent by the underlying GC tool. Whenever a view change occurs, a daemon prints out a list of members of the new view. Every message received is also displayed, together with its sender. On regular intervals whiteboard values (“LOAD”) of all group members are listed on the standard output by a separate thread. (See below for more details.)

The class `Test` sends a (`StringMessage`) message to the group, showing how the `multiCast()` works. To start the test, run:

```
java testjbora.Test
```

`Test` object joins the group, sends a message and leaves the group. At each step it prints to the standard output the information on what is happening, just like we saw in the case of daemons.

Details (Basic JBora features illustrated on this example):

Each daemon process is composed of two threads: (1) a main thread that executes the `receive()` operation and reacts to the received events; (2) a `WBHandler` thread that wakes up every 30 seconds to issue a `localCall()` passing an integer to the main thread; this integer is initially zero and then is the result received from the last `localCall()`; then, this thread writes the result from the `localCall()` on the whiteboard, reads the whiteboard value associated with each group member and prints these values on the standard output.

A `Daemon` class first instantiates a new `JBora` object and joins the group:

```
jbObject = new JBora(WB_WAIT, grpMembers, connName, grpName);  
jbObject.join();
```

The input parameters to the `JBora` constructor indicate, respectively, the waiting time before the whiteboard information is sent (by `multiCast()`) to other members, group size (expected number of group members, needed for determining the “primeness” of the View), name for this connection, and the name of the group to join.

Next, a whiteboard handler thread, **WBHandler** is started (with the same **JBora** object; All threads in a process must share the same **JBora** object).

```
WBHandler wbh = new WBHandler(jbObject);
Thread wbht = new Thread(wbh);
wbht.start();
```

We shall return to the **WBHandler** thread shortly. At this point, **Daemon** thread has finished its initialization process and may start receiving messages. It does this calling repeatedly the **JBora.receive()** method in an endless loop:

```
while(true) {
    evt = jbObject.receive();
    ...
}
```

The received event may be a message, a local message or a view change. If it is a message (a **String**) it will be displayed on the screen together with the sender id, and the load counter (number of messages received) will be increased by one. In a case it is a local message (from **WBHandler**), the load counter value (an integer) is sent back as the **localRespond()**:

```
if (evt.isLocalCall()) {
    Message msg=evt.getLocalMessage();
    IntMessage reply = new IntMessage(loadCounter);
    jbObject.localRespond(msg,reply);
}
```

Nota bene, a **localRespond()** contains both the message received and a reply to that message. If the message received is a view change, a daemon will print the identifiers of all members of the new view. Each group member is identified by its **ProcessID**.

```
else if (evt.isView()) {
    myView = evt.getView();
    ProcessID[] ps = myView.getMembers();
    ...
}
```

WBHandler thread every 30 seconds executes a **localCall()** to the main (**Daemon**) thread. The local message sent in this case is an integer (initially 0). If the respond message is of the expected type (also an integer, the load counter of the **Daemon**), the received value is set as the new whiteboard value:

```
msg = jbObject.localCall(new IntMessage(counter));
if (msg instanceof IntMessage) {
    IntMessage reply = (IntMessage)msg;
    counter=reply.getValue();
    jbObject.whiteBoardUpdate(counter);
    ...
}
```

After updating the whiteboard value for this **JBora** object, **WBHandler** prints out the whiteboard values of all members in the current view:

```
...
View currentView = jbObject.getCurrentView();
ProcessID[] ps = currentView.getMembers();
for (int i=0; i<ps.length; i++) {
    ProcessID pid = ps[i];
    int pidWbVal = jbObject.whiteBoardRead(pid);
    System.out.println(" LOAD: "+pid+ " > "+pidWbVal);
}
}
```

Then the example allows you to start additional processes that we simply called "tests". Each such process joins the group, waits for a couple of seconds, multicasts a simple message (String "Cheers!"), waits another couple of seconds, leaves the group and then terminates.

The integer that the main thread of each daemon returns in response to a **localCall()** is the number of messages received by that daemon so far. Thus, the whiteboard observed by each daemon simply corresponds to the number of tests that have been run since that daemon started.

The **Test** class instantiates the new **JBora** object and joins the group just like the **Daemon**. After a couple of seconds it sends (via **JBora.multiCast()**) a message (String) to other members of the group:

```
String mymessage="Cheers!";
StringMessage tm = new StringMessage(mymessage);
jbObject.multiCast(tm);
```

Please note that the member who sent the message to the group will also receive this message from the **JBora** GC layer. Only at this point the log "Message sent successfully" is displayed. After another couple of seconds, the test process will **leave** the group (and thus generate a new view change **Event** that the other members will **receive**). After leaving the group, the **Test** process terminates.

```
jbObject.leave();
```

The second example – Propagation:

This second example shows the use of the `JBora.propagate()` feature. To view this demo, you should start three (expected group size, default setting) `PropTest` processes either on a single VM or on different hosts:

```
java testjbora.PropTest
```

Each `PropTest` instantiates a new `JBora` object and joins the group. Then it starts a `receive()` loop until a view event of the expected group size is received. At this point, the `propagate()` method of the `JBora` object is invoked by each member. If the test is successful (propagation completed), each member prints out a list of members that have replied.

Please take a look at the source code (*PropTest.java*) for this example; it is quite simple and self-explanatory.