User Guide

Dominik Zindel
dominik [at] zindel.org
School of Computer Science
McGill University, Montreal, Canada

February 19, 2008

1 User Guide

This Section shall give a comprehensive guide for users of Postina. It both addresses the general use and some specific problems that might emerge when using Postina.

1.1 How To Use The Provided Version

This section explains how to use Postina with Pastry/Scribe. Postina is designed in a way that its use within another application is very simple. Only a few steps are necessary if you want to use Postina in another project:

1. Add the jar-files for Postina and Pastry to the classpath of your project.
2. Copy the files `postina.properties`, `log4.properties` and `freepastry.params` to a location in the classpath.
3. Adjust the parameters in `postina.properties` (see Section 1.4 for a more detailed description of properties).
4. Instantiate the class `ScribeNetworkLayer`.
5. Connect to the network using one of the three `connect`-methods in `ScribeNetworkLayer`. This will create the node and opens the necessary sockets. The operation may take a while. Note that the port defined in the properties has to be available. The method `connect` returns a `PostinaID` which you probably will want to store into a field.
6. If your application has to be informed about a dead client, register it to the `ScribeNetworkLayer` as a `PostinaClientListener`.
7. If you want Postina to inform your application about new messages, register it to the `ScribeNetworkLayer` as a `PostinaMsgListener`. Otherwise you will have to poll the queue yourself.

8. Enjoy Postina and use one of the many methods defined in the interface `PostinaNetworkLayer` by calling them on the `ScribeNetworkLayer`.

In case of troubles, Section 1.8 might provide useful hints. Note that Java 1.5 or newer is required to use Postina.

1.2 License

Postina has been released under the GNU Lesser General Public License. The license can be found on `http://www.gnu.org/licenses/`.

1.3 Multiple Network Interfaces

Many computers, especially servers, have two or more network interfaces. If the wrong network interface is chosen, the Pastry node is not visible to other nodes and the nodes cannot communicate with each other. However, in Pastry all nodes have to be accessible to all other nodes. That is, if the wrong interface is chosen, the Pastry network will not work. For technical reasons it is not possible to have a node listen on more than one interface[1], thus a sensible choice must be made.

By default, the IP address of the network interface is requested from the operating system automatically by Pastry. This means that the choice is random and not necessarily good, possibly leading to a broken system.

As an example, we consider the situation at the School of Computer Science at McGill University which is shown in Figure 1: There are three servers (`rogue`, `halo` and `oni`) that can be used as Pastry nodes. All three servers have two interfaces, an internal one which is used only for communication between these three servers (represented using red in the Figure) and an external one which is used for communication with all other clients (green). If the internal interface is chosen to bind to by Pastry, the three servers easily build a working network (red lines) but no other node can join this network. We have to use the external interface for the Pastry node (green lines, the three servers can also communicate over the external interface), the internal interface cannot be used.

1.3.1 Solution with Postina

It is not possible to let the application choose the right interface as there is no way to decide automatically which interface is accessible to other nodes. Therefore, this choice has to be made by a human.

The following procedure is used to decide which interface has to be chosen. The tests are given in decreasing priority order:

1. Address of the interface passed to the `connect` method as parameter.

[1] By definition, each node in Pastry is bound to an interface. Thus, two nodes would have to be build to listen on two interfaces and hence two 'rings' would be created. MultiRing would have to be used to communicate between these two networks.

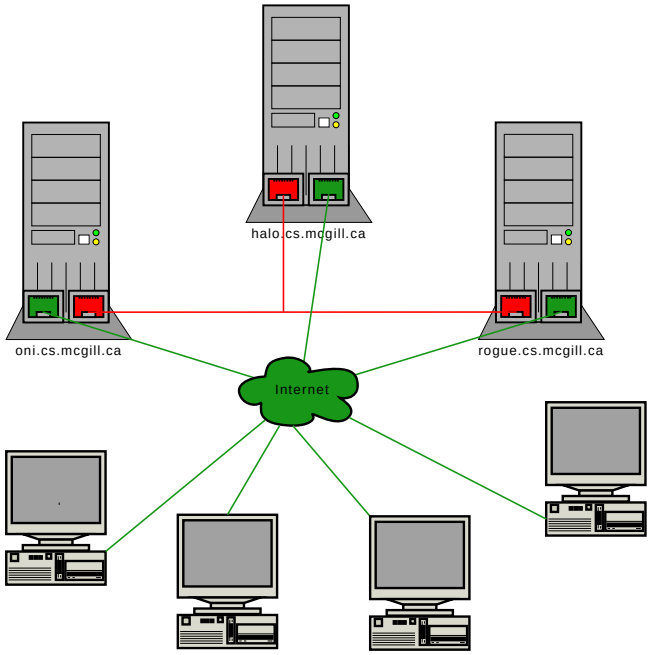

Figure 1: The topology of the game research servers.

2. Address of the interface defined as property `localBindAddress` in a property file (see Section 1.4).

3. Address of the interface returned by the operating system.

These procedure is guaranteed to work if there is a network interface present.

1.4 Properties

Postina uses the `Properties` of Java and looks out for two different files in the classpath: the default properties stored in a file called `postina.properties` and local properties defined in a file containing the hostname in lowercases in its name. The exact format of the local file name is `postina.<hostname>.properties` where <hostname> is the fully qualified hostname in lowercases of the machine for which these properties should be applied. Example: `postina.se-6.cs.mcgill.ca.properties`

The default properties can be overridden by defining the property in the local property file as a local property has higher priority than a default property.

1.4.1 Format

As the Java properties are used in Postina, the property files have to comply with the format defined in the Java class `Properties`. Each line has to contain a

single property which always has the format `propertyName = propertyValue`. Please see the paragraph below for a complete list of possible properties with possible values.

1.4.2 Available Properties

This paragraph gives a complete list of all properties available together with a short description of each property:

bootAddresses The bootstrap nodes to be used when joining the network. *Format:* comma-separated list of `IP:Port`. *Example:* `132.206.3.142:8899,132.206.3.140:8899,132.206.3.141:8899`

localBindAddress The address of the network interface to which Postina should bind. To be used when a machine has multiple interfaces. *Format:* IP address. *Example:* `132.206.51.88`

localBindPort The port that should be used by Postina for the network communication. *Format:* Integer value. *Example:* `8899`

reliability.timeout The number of milliseconds before a message times out. *Format:* Long value. *Example:* `40000`

reliability.maxNumTries The number of times Postina should attempt to resend a message before giving up. *Format:* Integer value. *Example:* `4`

reliability.fatalNumLostMsg The number of definitely lost messages that are allowed before a client is declared dead. *Format:* Integer value. *Example:* `10`

1.5 Upgrade Pastry/Scribe

To replace the version of FreePastry by a newer one, the jar-file of Pastry simply has to be replaced by a new one.

1.6 How To Replace Pastry/Scribe

Although Postina comes with Pastry/Scribe by default, developers are free to choose another underlying system. As the API of Postina is well separated from the effective implementation, switching to another system is very easy. The new system simply has to implement the interfaces defined by Postina as shown in Figure 2 on page 5.

In terms of the Java code, all interfaces defined in the package `ca.mcgill.cs.postina` have to be implemented by the new system. Additionally, the package `ca.mcgill.cs.postina.util.properties` has to be kept as it is and the format and naming of the property files has to remain unaffected. In other words, the only code that has to be replaced to use another system is the package `ca.mcgill.cs.postina.scribe`.

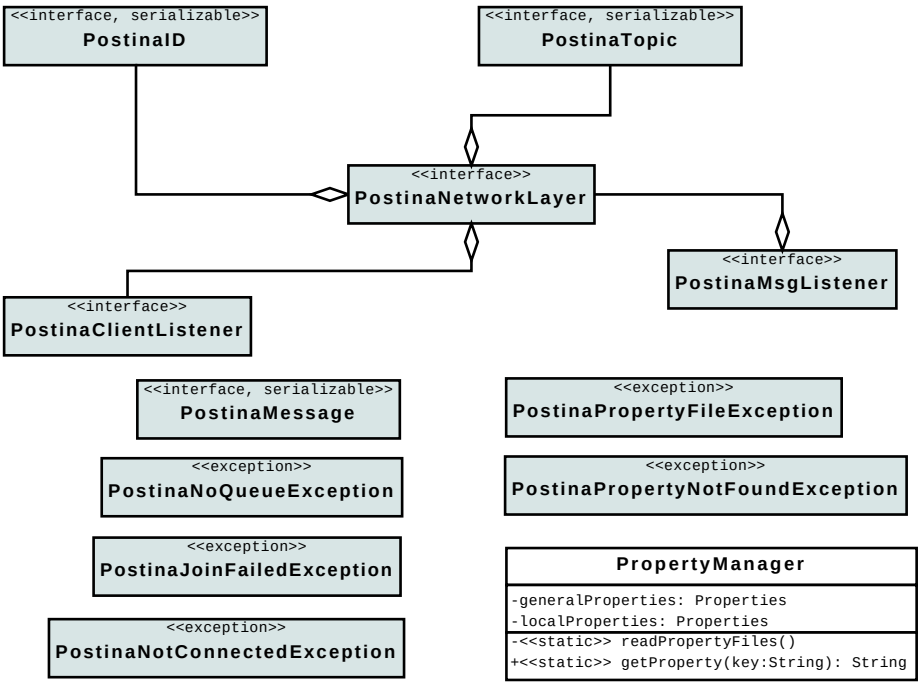

Figure 2: The key interfaces of Postina.

1.7 Logging

Postina uses `log4j` for logging. By default, logging messages of level `INFO` or higher are printed on the standard output, that is on the terminal. This setting can be changed in the configuration file `log4j.properties` which has to be added to the classpath.

1.8 Trouble Shooting

This section gives some important, possibly time-saving, hints to the users of Postina:

- If multiple nodes of Postina are run on the same computer, the same properties are used. Thus, each node tries to bind to the same port. As this is not possible, Postina using Pastry/Scribe automatically binds to the next free port it can find. The used port is indicated in the message printed out when starting up Postina ("Finished creating new node:").
- The local properties override the global properties, that is if a change in the properties does not have any effect, the same property might be defined in a machine-specific file. The use of the properties is explained in Section 1.4.
- If you are using Postina in a network using Network Address Translation (NAT) or firewalls please read Section 6.1 of the complete report.