Mobicents JAIN SLEE SIP11 Resource Adaptor User Guide

by Bartosz Baranowski and Eduardo Martins

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts* [https://fedorahosted.org/liberation-fonts/] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight key caps and key-combinations. For example:

> To see the contents of the file `my_next_bestselling_novel` in your current working directory, enter the `cat my_next_bestselling_novel` command at the shell prompt and press **Enter** to execute the command.

The above includes a file name, a shell command and a key cap, all presented in Mono-spaced Bold and all distinguishable thanks to context.

Key-combinations can be distinguished from key caps by the hyphen connecting each part of a key-combination. For example:

> Press **Enter** to execute the command.
>
> Press **Ctrl**+**Alt**+**F1** to switch to the first virtual terminal. Press **Ctrl**+**Alt**+**F7** to return to your X-Windows session.

The first sentence highlights the particular key cap to press. The second highlights two sets of three key caps, each set pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in `Mono-spaced Bold`. For example:

> File-related classes include `filesystem` for file systems, `file` for files, and `dir` for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialogue box text; labelled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System > Preferences > Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications > Accessories > Character Map** from the main menu bar. Next, choose **Search > Find** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit > Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in Proportional Bold and all distinguishable by context.

Note the **>** shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow 'Select **Mouse** from the **Preferences** sub-menu in the **System** menu of the main menu bar' approach.

`Mono-spaced Bold Italic` or ***Proportional Bold Italic***

Whether Mono-spaced Bold or Proportional Bold, the addition of Italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type `ssh` *`username@domain.name`* at a shell prompt. If the remote machine is `example.com` and your username on that machine is john, type `ssh john@example.com`.
>
> The `mount -o remount` *`file-system`* command remounts the named file system. For example, to remount the `/home` file system, the command is `mount -o remount /home`.
>
> To see the version of a currently installed package, use the `rpm -q` *`package`* command. It will return a result as follows: *`package-version-release`*.

Note the words in bold italics above username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> When the Apache HTTP Server accepts requests, it dispatches child processes or threads to handle them. This group of child processes or threads is known as

a *server-pool*. Under Apache HTTP Server 2.0, the responsibility for creating and maintaining these server-pools has been abstracted to a group of modules called *Multi-Processing Modules* (*MPMs*). Unlike other modules, only one module from the MPM group can be loaded by the Apache HTTP Server.

1.2. Pull-quote Conventions

Two, commonly multi-line, data types are set off visually from the surrounding text.

Output sent to a terminal is set in `Mono-spaced Roman` and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in `Mono-spaced Roman` but are presented and highlighted as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
  public static void main(String args[])
      throws Exception
  {
    InitialContext iniCtx = new InitialContext();
    Object         ref    = iniCtx.lookup("EchoBean");
    EchoHome       home   = (EchoHome) ref;
    Echo           echo   = home.create();

    System.out.println("Created Echo");

    System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
  }

}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

A note is a tip or shortcut or alternative approach to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring Important boxes won't cause data loss but may cause irritation and frustration.

Warning

A Warning should not be ignored. Ignoring warnings will most likely cause data loss.

2. Provide feedback to the authors!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in the the *Issue Tracker* [http://code.google.com/p/mobicents/issues/list], against the product **Mobicents JAIN SLEE SIP11 Resource Adaptor**, or contact the authors.

When submitting a bug report, be sure to mention the manual's identifier: JAIN_SLEE_SIP11_RA_User_Guide

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Introduction to Mobicents JAIN SLEE SIP11 Resource Adaptor

This resource adaptor provides a SIP API for JAIN SLEE applications, adapting the JAIN SIP 1.2 specification. JAIN SIP 1.2 is a Java specification for the Session Initiation Protocol, as defined by the protocol specs done by the IETF. Both transaction and dialog layers of the SIP protocol are available, to support all types of SIP applications through a single API. Lower level applications, such as SIP Proxies or Registrars typically use the transaction layer exclusively, while UAC, UAS and B2BUA higher level SIP applications rely on the dialog layer.

Events represent SIP messages received by the SIP stack, or failure use cases such as timeouts. Unlike the base JAIN SIP 1.2 API, SIP Requests with different SIP methods are fired as different event types, the same happens for SIP Responses with status code. The events are fired on transaction or dialog activities.

The Activities are the SIP Transactions and Dialogs, which applications in the SLEE may use to send SIP Requests and Responses, and to receive the events related with incoming messages.

Resource Adaptor Type

The JAIN SIP 1.2 Resource Adaptor Type is specified in Appendix D of the JAIN SLEE 1.1 Specification. The specification can be freely downloaded from *http://jcp.org/aboutJava/communityprocess/final/jsr240/index.html* and includes Sbb code examples.

Resource Adaptor Implementation

The RA implementation uses the Mobicents JAIN SIP HA stack, an extension of the JAIN SIP Reference Implementation which provides high availability and fault tolerance. The stack is the result of the work done by Mobicents JAIN SLEE and SIP Servlets development teams, and source code is provided in all releases.

3.1. Configuration

The Resource Adaptor supports configuration only at Resource Adaptor Entity creation time, the following table enumerates the configuration properties:

Table 3.1. Resource Adaptor's Configuration Properties

Property Name	Description	Property Type	Default Value
javax.sip.IP_ADDRESS	the IP address to which the SIP stack should attach - if value is not specified the RA will use the underlying Java EE container's bind address address	java.lang.String	
javax.sip.OUTBOUND_PROXY	the outbound proxy of the SIP Stack. The format for this string is "ipaddress:port/transport" i.e. 129.1.22.333:5060/UDP. This property is optional	java.lang.String	
javax.sip.PORT	the port to which the SIP stack should listen	java.lang.Integer	5060
javax.sip.TRANSPORT	the list of supported transports, separated with ","	java.lang.String	UDP
org.mobicents.ha.javax.sip.BALANCERS	the LIST OF SIP balancers, in the form of "HOST:PORT", separated by ";", it is only used if the heart beat service property is defined	java.lang.String	

Property Name	Description	Property Type	Default Value
org.mobicents.ha.javax.sip.LoadBalancerHeartBeatingServiceClassName	class responsible for the heart beats exchanged with the platform's SIP Balancer - if not specified the JAIN SIP HA stack won't use such feature	java.lang.String	

Important

JAIN SLEE 1.1 Specification requires values set for properties without a default value, which means the configuration for those properties are mandatory, otherwise the Resource Adaptor Entity creation will fail!

3.2. Default Resource Adaptor Entities

There is a single Resource Adaptor Entity created when deploying the Resource Adaptor, named `SipRA`. The `SipRA` entity uses the default Resource Adaptor configuration, specified in *Section 3.1, "Configuration"*.

The `SipRA` entity is also bound to Resource Adaptor Link Name `SipRA`, to use it in an Sbb add the following XML to its descriptor:

```
<resource-adaptor-type-binding>

   <resource-adaptor-type-ref>
      <resource-adaptor-type-name>
         JAIN SIP
      </resource-adaptor-type-name>
      <resource-adaptor-type-vendor>
         javax.sip
      </resource-adaptor-type-vendor>
      <resource-adaptor-type-version>
         1.2
      </resource-adaptor-type-version>
   </resource-adaptor-type-ref>

   <activity-context-interface-factory-name>
```

```
        slee/resources/jainsip/1.2/acifactory
    </activity-context-interface-factory-name>

    <resource-adaptor-entity-binding>
      <resource-adaptor-object-name>
        slee/resources/jainsip/1.2/provider
      </resource-adaptor-object-name>
      <resource-adaptor-entity-link>
        SipRA
      </resource-adaptor-entity-link>
    </resource-adaptor-entity-binding>

  </resource-adaptor-type-binding>
```

3.3. Traces and Alarms

3.3.1. Tracers

Each Resource Adaptor Entity uses a single JAIN SLEE 1.1 Tracer, named `SipResourceAdaptor`.

3.3.2. Alarms

No alarms are set by this Resource Adaptor.

Setup

4.1. Pre-Install Requirements and Prerequisites

Ensure that the following requirements have been met before continuing with the install.

4.1.1. Hardware Requirements

The Resource Adaptor hardware's main concern is RAM memory and Java Heap size, the more the better. For instance, while the underlying Mobicents JAIN SLEE may run with 1GB of RAM, 8GB is needed to achieve performance higher than 400 new calls per second.

Of course, memory is only needed to store the Resource Adaptor state, the faster the CPU more calls per second are supported, yet no particular CPU is a real requirement to use the RA.

4.1.2. Software Prerequisites

The RA requires Mobicents JAIN SLEE properly set.

4.2. Mobicents JAIN SLEE SIP11 Resource Adaptor Source Code

4.2.1. Release Source Code Building

1. **Downloading the source code**

Important

Subversion is used to manage its source code. Instructions for using Subversion, including install, can be found at *http://svnbook.red-bean.com*

Use SVN to checkout a specific release source, the base URL is http://mobicents.googlecode.com/svn/tags/servers/jain-slee/2.x.y/resources/sip11, then add the specific release version, lets consider 2.0.0.CR1.

```
[usr]$ svn co http://mobicents.googlecode.com/svn/tags/servers/jain-slee/2.x.y/resources/sip11/2.0.0.CR1 slee-ra-sip11-2.0.0.CR1
```

2. **Building the source code**

Important

Maven 2.0.9 (or higher) is used to build the release. Instructions for using Maven2, including install, can be found at *http://maven.apache.org*

Use Ant to build the binary.

```
[usr]$ cd slee-ra-sip11-2.0.0.CR1
[usr]$ mvn install
```

Once the process finishes you should have the `deployable-unit` jar file in the `target` directory, if Mobicents JAIN SLEE is installed and environment variable JBOSS_HOME is pointing to its underlying JBoss Application Server directory, then the deployable unit jar will also be deployed in the container.

4.2.2. Development Trunk Source Building

Similar process as for *Section 4.2.1, "Release Source Code Building"*, the only change is the SVN source code URL, which is http://mobicents.googlecode.com/svn/trunk/servers/jain-slee/resources/sip11.

4.3. Installing Mobicents JAIN SLEE SIP11 Resource Adaptor

To install the Resource Adaptor simply execute provided ant script `build.xml` default target:

```
[usr]$ ant
```

The script will copy the RA deployable unit jar to the `default` Mobicents JAIN SLEE server profile deploy directory, to deploy to another server profile use the argument `-Dnode=`.

4.4. Uninstalling Mobicents JAIN SLEE SIP11 Resource Adaptor

To uninstall the Resource Adaptor simply execute provided ant script `build.xml undeploy` target:

```
[usr]$ ant undeploy
```

The script will delete the RA deployable unit jar from the `default` Mobicents JAIN SLEE server profile deploy directory, to undeploy from another server profile use the argument `-Dnode=`.

Clustering

5.1. Failover

The SIP Stack used by the RA supports `ESTABLISHED SIP DIALOG` failover. This means that an application must be in charge of properly adapting its state machine, to recover SIP transaction or early dialogs failures, once message retransmissions are received.

5.2. Load Balancing

The RA can be used with Mobicents SIP Load Balancer. The recommended version is 1.0.BETA12.

5.2.1. Configuring the Resource Adaptor to be used with Mobicents SIP Load Balancer

There are three properties which define how the RA connects to Mobicents SIP Load Balancer:

org.mobicents.ha.javax.sip.BALANCERS
: the property must be configured with the list of load balancer IP address and internal ports. As an example, suppose a single Mobicents SIP Load Balancer is running with IP `192.168.0.1` and internal port `5065`, the property would be set with value `192.168.0.1:5065`. To specify multiple balancers use `;` as separator.

org.mobicents.ha.javax.sip.LoadBalancerHeartBeatingServiceClassName
: this property is optional, defines the class name of the HeartBeating service implementation, currently the only one available is `org.mobicents.ha.javax.sip.LoadBalancerHeartBeatingServiceImpl`

org.mobicents.ha.javax.sip.LoadBalancerElector
: this property is optional, defines the class of the load balancer elector from JAIN SIP HA Stack. The elector is used to define which load balancer will receive outgoing requests, which are out of dialog or in dialog with null state. Currently only one elector implementation is available, `org.mobicents.ha.javax.sip.RoundRobinLoadBalancerElector`, which, as the class name says, uses round robin algorythm to select the balancer.

5.2.2. Mobicents SIP Load Balancer

The Mobicents SIP load balancer is used to balance the load of SIP service requests and responses between nodes in a JAIN SLEE cluster, increasing the performance and availability of SIP services and applications.

In terms of functionality, the Mobicents SIP Load Balancer is a simple stateless proxy server that intelligently forwards SIP session requests and responses between User Agents (UAs) on a

Wide Area Network (WAN), and SIP RA nodes, which are almost always located on a Local Area Network (LAN). All SIP requests and responses pass through the SIP load balancer.

5.2.2.1. SIP Load Balancing Basics

All User Agents send SIP messages, such as `INVITE` and `MESSAGE`, to the same SIP URI (the IP address and port number of the SIP load balancer on the WAN). The load balancer then parses, alters, and forwards those messages to an available node in the cluster. If the message was sent as a part of an existing SIP session, it will be forwarded to the cluster node which processed that User Agent's original transaction request.

The SIP RA that receives the message acts upon it and sends a response back to the SIP load balancer. The SIP load balancer reparses, alters and forwards the message back to the original User Agent. This entire proxying and provisioning process is carried out independent of the User Agent, which is only concerned with the SIP service or application it is using.

By using the load balancer, SIP traffic is balanced across a pool of available SIP RAs, increasing the overall throughput of the SIP service or application running on either individual nodes of the cluster.

The SIP load balancer is also able to fail over requests mid-call from unavailable nodes to available ones, thus increasing the reliability of the SIP service or application. The load balancer increases throughput and reliability by dynamically provisioning SIP service requests and responses across responsive nodes in a cluster. This enables SIP applications to meet the real-time demand for SIP services.

5.2.2.2. Pluggable Balancer Algorithms

The SIP load balancer exposes an interface to allow users to customize the routing decision algorithm. Only one algorithm is active at any time and it is specified with the `algorithmClass` property in the configuration file.

It is completely up to the algorithm how and whether to support distributed architecture or how to store the information needed for session affinity. The algorithms will be called for every SIP request and other significant events to make proper routing decisions.

The following is a list of the built-in algorithms:

org.mobicents.tools.sip.balancer.CallIDAffinityBalancerAlgorithm
: This algorithm does not support distributed use case. It selects nodes randomly to serve a give Call-ID extracted from the requests and responses. It keeps a map with `Call-ID -> nodeId` associations and this map is not shared with other load balancers which will cause them to make different decisions.

org.mobicents.tools.sip.balancer.HeaderConsistentHashBalancerAlgorithm
: This algorithm can be used in distributed load balancer configurations. It extracts the hash value of specific headers from SIP messages to decide which application server node will

handle the request. Information about the options in this algorithms is available in the balancer configuration file comments.

org.mobicents.tools.sip.balancer.PersistentConsistentHashBalancerAlgorithm
: This algorithm can be used in distributed load balancer configurations. It is similar to the previous algorithm, but it attempts to keep session affinity even when the cluster nodes are removed or added, which would normally cause hash values to point to different nodes.

5.2.2.3. Distributed load balancing

When the capacity of a single load balancer is exceeded, multiple load balancers can be used. With the help of an IP load balancer the traffic can be distributed between all SIP load balancers based on some IP rules or round-robin. With consistent hash and `jvmRoute`-based balancer algorithms it doesn't matter which SIP load balancer will process the request, because they would all make the same decisions based on information in the requests (headers, parameters or cookies) and the list of available nodes. With consistent hash algorithms there is no state to be preserved in the SIP balancers.

5.2.2.4. Implementation of the Mobicents SIP Load Balancer

Each individual Mobicents JAIN SLEE SIP RA in the cluster is responsible for contacting the SIP load balancer and relaying its health status and regular "heartbeats".

From these health status reports and heartbeats, the SIP load balancer creates and maintains a list of all available and healthy nodes in the cluster. The load balancer forwards SIP requests between these cluster nodes, providing that the provisioning algorithm reports that each node is healthy and is still sending heartbeats.

If an abnormality is detected, the SIP load balancer removes the unhealthy or unresponsive node from the list of available nodes. In addition, mid-session and mid-call messages are failed over to a healthy node.

The SIP load balancer first receives SIP requests from endpoints on a port that is specified in its Configuration Properties configuration file. The SIP load balancer, using a round-robin algorithm, then selects a node to which it forwards the SIP requests. The load balancer forwards all same-session requests to the first node selected to initiate the session, providing that the node is healthy and available.

5.2.2.5. SIP Message Flow

The Mobicents SIP load balancer appends itself to the `Via` header of each request, so that returned responses are sent to the SIP Balancer before they are sent to the originating endpoint.

The load balancer also adds itself to the path of subsequent requests by adding Record-Route headers. It can subsequently handle mid-call failover by forwarding requests to a different node in the cluster if the node that originally handled the request fails or becomes unavailable. The SIP load balancer immediately fails over if it receives and unhealthy status, or irregular heartbeats from a node.

5.2.2.6. SIP Load Balancer: Installing, Configuring and Running

The load balancer can be downloaded from `http://repository.jboss.org/maven2/org/mobicents/tools/sip-balancer/1.0.BETA12`. There you will find the balancer's executable jar with dependencies (`sip-balancer-1.0.BETA12-jar-with-dependencies.jar`) , along with javadocs and sources jars.

5.2.2.6.1. Configuring the Mobicents SIP Load Balancer

Configuration is done through a properties file which path is then passed as argument. Below is a configuration properties file example:

```
# The binding address of the load balancer
host=127.0.0.1

# The RMI port used for heartbeat signals
rmiRegistryPort=2000

# The SIP port used where client should connect
externalPort=5060

# The SIP port from where servers will receive messages
# delete if you want to use only one port for both inbound and outbound)
# if you like to activate the integrated HTTP load balancer, this is the entry point
internalPort=5065

# The HTTP port for HTTP forwarding
httpPort=2080

#Specify UDP or TCP (for now both must be the same)
internalTransport=UDP
externalTransport=UDP

# If you are using IP load balancer, put the IP address and port here
#externalIpLoadBalancerAddress=127.0.0.1
#externalIpLoadBalancerPort=111

# Requests initited from the App Servers can route to this address (if you are using 2 IP load
 balancers for bidirectional SIP LB)
#internalIpLoadBalancerAddress=127.0.0.1
#internalIpLoadBalancerPort=111

# Designate extra IP addresses as serer nodes
#extraServerNodes=222.221.21.12:21,45.6.6.7:9003,33.5.6.7,33.9.9.2
```

```
# Call-ID affinity algortihm settings. This algorithm is the default. No need to uncomment it.
#algorithmClass=org.mobicents.tools.sip.balancer.CallIDAffinityBalancerAlgorithm
# This property specifies how much time to keep an association before being evitcted.
# It is needed to avoid memory leaks on dead calls. The time is in seconds.
#callIdAffinityMaxTimeInCache=500

# Uncomment to enable the consistent hash based on Call-ID algorithm.
#algorithmClass=org.mobicents.tools.sip.balancer.HeaderConsistentHashBalancerAlgorithm
# This property is not required, it defaults to Call-ID if not set, cna be "from.user" or "to.user" when
 you want the SIP URI username
#sipHeaderAffinityKey=Call-ID
#specify the GET HTTP parameter to be used as hash key
#httpAffinityKey=appsession

# Uncomment to enable the persistent consistent hash based on Call-ID algorithm.
#algorithmClass=org.mobicents.tools.sip.balancer.PersistentConsistentHashBalancerAlgorithm
# This property is not required, it defaults to Call-ID if not set
#sipHeaderAffinityKey=Call-ID
#specify the GET HTTP parameter to be used as hash key
#httpAffinityKey=appsession

#This is the JBoss Cache 3.1 configuration file (with jgroups), if not specified it will use default
#persistentConsistentHashCacheConfiguration=/home/config.xml

# Call-ID affinity algortihm settings. This algorithm is the default. No need to uncomment it.
#algorithmClass=org.mobicents.tools.sip.balancer.CallIDAffinityBalancerAlgorithm
# This property specifies how much time to keep an association before being evitcted.
# It is needed to avoid memory leaks on dead calls. The time is in seconds.
#callIdAffinityMaxTimeInCache=500

# Uncomment to enable the consistent hash based on Call-ID algorithm.
#algorithmClass=org.mobicents.tools.sip.balancer.HeaderConsistentHashBalancerAlgorithm
# This property is not required, it defaults to Call-ID if not set, cna be "from.user" or "to.user" when
 you want the SIP URI username
#sipHeaderAffinityKey=Call-ID
#specify the GET HTTP parameter to be used as hash key
#httpAffinityKey=appsession

# Uncomment to enable the persistent consistent hash based on Call-ID algorithm.
#algorithmClass=org.mobicents.tools.sip.balancer.PersistentConsistentHashBalancerAlgorithm
# This property is not required, it defaults to Call-ID if not set
#sipHeaderAffinityKey=Call-ID
#specify the GET HTTP parameter to be used as hash key
```

```
#httpAffinityKey=appsession

#This is the JBoss Cache 3.1 configuration file (with jgroups), if not specified it will use default
#persistentConsistentHashCacheConfiguration=/home/config.xml

#JSIP stack configuration.....
javax.sip.STACK_NAME = SipBalancerForwarder
javax.sip.AUTOMATIC_DIALOG_SUPPORT = off
// You need 16 for logging traces. 32 for debug + traces.
// Your code will limp at 32 but it is best for debugging.
gov.nist.javax.sip.TRACE_LEVEL = 32
gov.nist.javax.sip.DEBUG_LOG = logs/sipbalancerforwarderdebug.txt
gov.nist.javax.sip.SERVER_LOG = logs/sipbalancerforwarder.xml
gov.nist.javax.sip.THREAD_POOL_SIZE = 64
gov.nist.javax.sip.REENTRANT_LISTENER = true
```

An overview of most important properties:

host
: Local IP address, or interface, on which the SIP load balancer will listen for incoming requests.

externalPort
: Port on which the SIP load balancer listens for incoming requests from SIP User Agents.

internalPort
: Port on which the SIP load balancer forwards incoming requests to available, and healthy, SIP Servlets Server cluster nodes.

rmiRegistryPort
: Port on which the SIP load balancer will establish the RMI heartbeat connection to the application servers. When this connection fails or a disconnection instruction is received, an application server node is removed and handling of requests continues without it by redirecting the load to the lie nodes.

internalTransport
: Transport protocol for the internal SIP connections associated with the internal SIP port of the load balancer. Possible choices are `UDP`, `TCP` and `TLS`.

externalTransport
: Transport protocol for the external SIP connections associated with the external SIP port of the load balancer. Possible choices are `UDP`, `TCP` and `TLS`. It must match the transport of the internal port.

externalIpLoadBalancerAddress
: Address of the IP load balancer (if any) used for incoming requests to be distributed in the direction of the application server nodes. This address may be used by the SIP load balancer to be put in SIP headers where the external address of the SIP load balancer is needed.

externalIpLoadBalancerPort
: The port of the external IP load balancer. Any messages arriving at this port should be distributed across the external SIP ports of a set of SIP load balancers.

internalIpLoadBalancerAddresst
: Address of the IP load balancer (if any) used for outgoing requests (requests initiated from the servers) to be distributed in the direction of the clients. This address may be used by the SIP load balancer to be put in SIP headers where the internal address of the SIP load balancer is needed.

internalIpLoadBalancerPort
: The port of the internal IP load balancer. Any messages arriving at this port should be distributed across the internal SIP ports of a set of SIP load balancers.

extraServerNodes
: Comma-separated list of hosts that are server nodes. You can put here alternative names of the application servers and they will be recognized. Names are important, because they might be used for direction-analysis. Requests coming from these server will go in the direction of the clients and will not be routed back to the cluster.

algorithmClass
: The fully-qualified Java class name of the balancing algorithm to be used. There are three algorithms to choose from and you can write your own to implement more complex routing behaviour. Refer to the sample configuration file for details about the available options for each algorithm. Each algorithm can have algorithm-specific properties for fine-grained configuration.

Important

The remaining keys and properties in the configuration properties file can be used to tune the JAIN SIP stack, but are not specifically required for load balancing. To assist with tuning, a comprehensive list of implementing classes for the SIP Stack is available from the *Interface SIP Stack page on nist.gov* [http://snad.ncsl.nist.gov/proj/iptel/jain-sip-1.2/javadoc/javax/sip/SipStack.html]. For a comprehensive list of properties associated with the SIP Stack implementation, refer to *Class SipStackImpl page on nist.gov* [http://snad.ncsl.nist.gov/proj/iptel/jain-sip-1.2/javadoc/gov/nist/javax/sip/SipStackImpl.html].

5.2.2.6.2. Running Mobicents SIP Balancer

Start the SIP load balancer, ensuring the Configuration Properties file (`lb.properties` in this example) is specified. In the Unix terminal, or using the Windows Command Prompt, the SIP Load Balancer is started by issuing a command similar to this one:

```
java -jar sip-balancer-1.0.BETA12-jar-with-dependencies.jar -mobicents-
balancer-config=lb-configuration.properties
```

Executing the SIP load balancer produces output similar to the following example:

```
[user]$ java -jar sip-balancer-1.0.BETA12-jar-with-dependencies.jar -
mobicents-balancer-config=lb-configuration.properties
Oct 21, 2008 1:10:58 AM
 org.mobicents.tools.sip.balancer.SIPBalancerForwarder start
INFO: Sip Balancer started on address 127.0.0.1, external port : 5060,
 port : 5065
Oct 21, 2008 1:10:59 AM org.mobicents.tools.sip.balancer.NodeRegisterImpl
 startServer
INFO: Node registry starting...
Oct 21, 2008 1:10:59 AM org.mobicents.tools.sip.balancer.NodeRegisterImpl
 startServer
INFO: Node expiration task created
Oct 21, 2008 1:10:59 AM org.mobicents.tools.sip.balancer.NodeRegisterImpl
 startServer
INFO: Node registry started
```

The output shows the IP address on which the SIP load balancer is listening, as well as the external and internal listener ports.

5.2.2.6.3. Stopping

Assuming that you started the load balancer as a foreground process in the OS terminal, the easiest way to stop it is by pressing the **Ctrl**+**C** key combination in the same terminal in which you started it.

This should produce similar output to the following:

```
^COct 21, 2008 1:11:57 AM
 org.mobicents.tools.sip.balancer.SipBalancerShutdownHook run
INFO: Stopping the sip forwarder
```

5.2.2.6.4. Uninstalling

To uninstall the SIP load balancer, delete the JAR file you installed.

Appendix A. Revision History

Revision History

Revision 1.0	Tue Dec 30 2009	EduardoMartins

Creation of the Mobicents JAIN SLEE SIP11 RA User Guide.

Index

F