



eLiveServer™

Server Developer Guide

Book version 2.5.2

Table of Contents

1	ELIVEENGINE™ ARCHITECTURE OVERVIEW	4
2	ELIVESERVER™: NETWORK TOPOLOGY OVERVIEW	5
2.1	Installation on a shared server	5
2.2	Installation on a dedicated server	6
2.3	Login and integration with website	6
3	ELIVESERVER PROTOCOL	7
3.1	eLiveServer™ Server to Client protocol	7
3.2	eLiveServer™ Client to Server protocol	7
4	ELIVESERVER BOT API	9
4.1	What is an eLiveServer Bot?	9
4.2	Java Interface to create a bot	9
4.3	Sample Bot	9
4.4	Testing the plugin with eLiveServer™	10
5	ELIVESERVER™ PLUGIN API	11
5.1	What is an eLiveServer™ plugin?	11
5.2	Java interface to create a plugin	11
5.3	Testing the plugin with eLiveServer™	12
5.4	Optional translation plugin	12
6	ELIVESERVER™ HTTP API	14
6.1	API calls	14
6.2	API List	14
6.2.1	Introduction	14
6.2.2	Number of people in one chat-room	14
6.2.3	List of users in a chat-room	14
6.2.4	List of chat-rooms	15
6.2.5	Number of chat-rooms	15
6.2.6	Number of people in all chat-rooms	15
6.2.7	Status of user: online/offline	15
6.2.8	Send a message to a room	15
6.2.9	Send a private message	15
6.2.10	Stop server	15
6.2.11	General statistics about server	15

6.2.12	License information	16
7	FAIL OVER MANAGEMENT	17
7.1	Chat Client fail over feature	17
7.2	ELiveServer™	17
7.2.1	Process runner tool	17
7.2.2	Monitoring	18
8	WEB SERVER INTEGRATION	19
9	PROXY AND FIREWALL SUPPORT	20
9.1	Introduction about HTTP tunneling	20
9.2	Apache Proxy module	20
9.3	HTTPconnector Servlet	21
10	DATABASE INTEGRATION	23
11	BANDWIDTH FOR SERVER	25
12	SCALABILITY AND ADVANCED SET-UP	26
12.1	Number of concurrent users	26
12.2	Quality of chat service	26
12.3	Advanced tuning with Linux server Kernel 2.2.x	26
13	ELIVESERVER™ AND INSTANT MESSAGING	28

1 EliveEngine™ architecture overview

EliveEngine™ is composed

- by two core engines:
 - EliveServer™: server for realtime communication applications. EliveServer™ is the core engine of solutions: 12Planet ELiveServer™, 12Planet AgentServer™, 12Planet MessengerServer™
 - EliveWeb™: framework for content management and login features. ELiveWeb™ is the core engine for 12Planet ForumServer™
- and modules build on these engines. It is possible to build custom modules

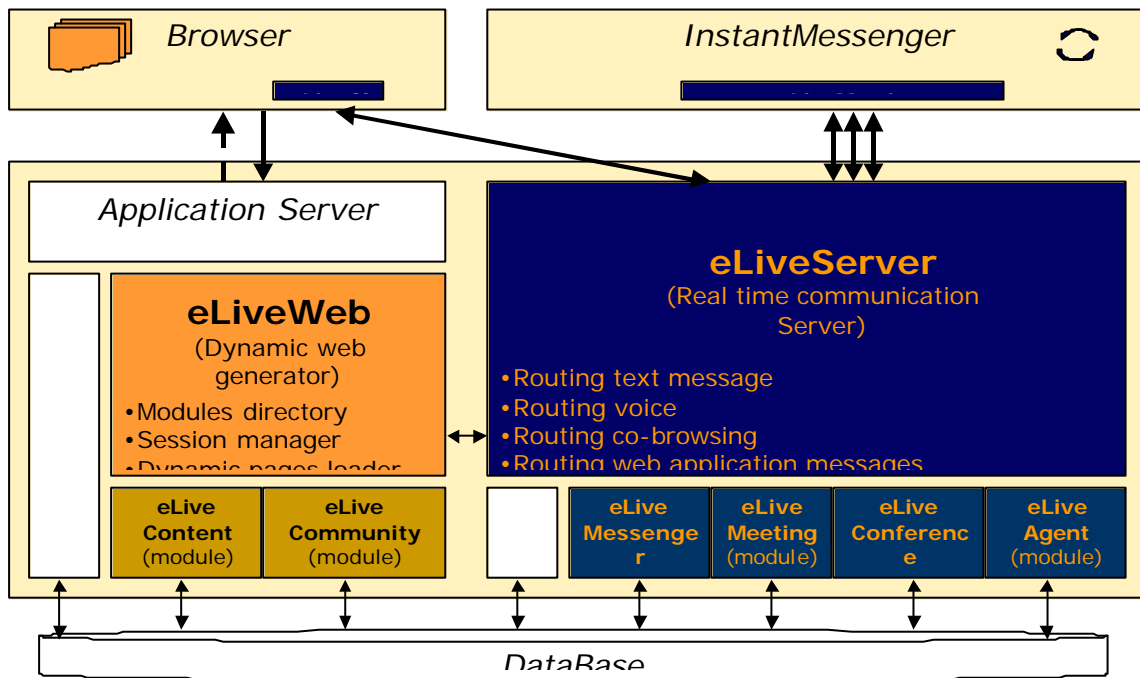
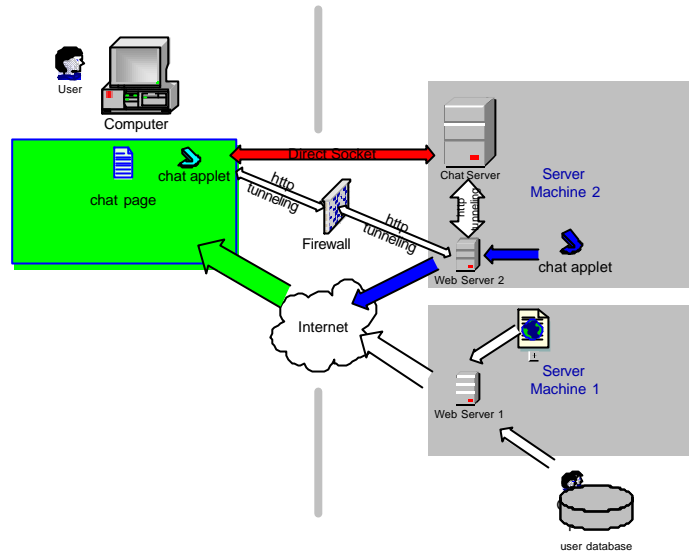


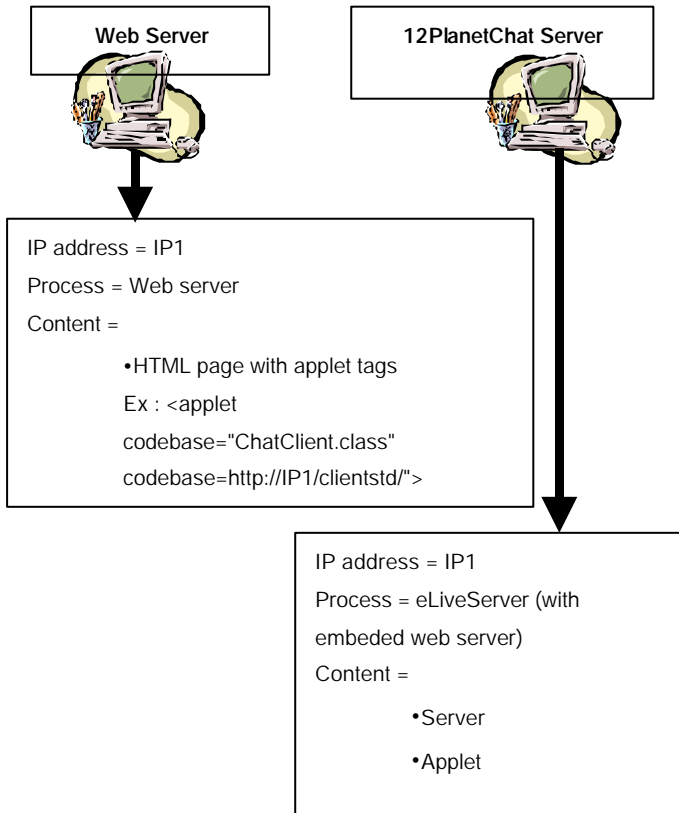
Figure 1: eLiveEngine™ Architecture Overview

2 eLiveServer™: network topology overview



2.1 Installation on a shared server

12Planet eLiveServer™ is installed on the same machine as your web server.



The eLiveServer™ is a software that you will install on the same server machine as your Web Server software (Apache, IIS ...)

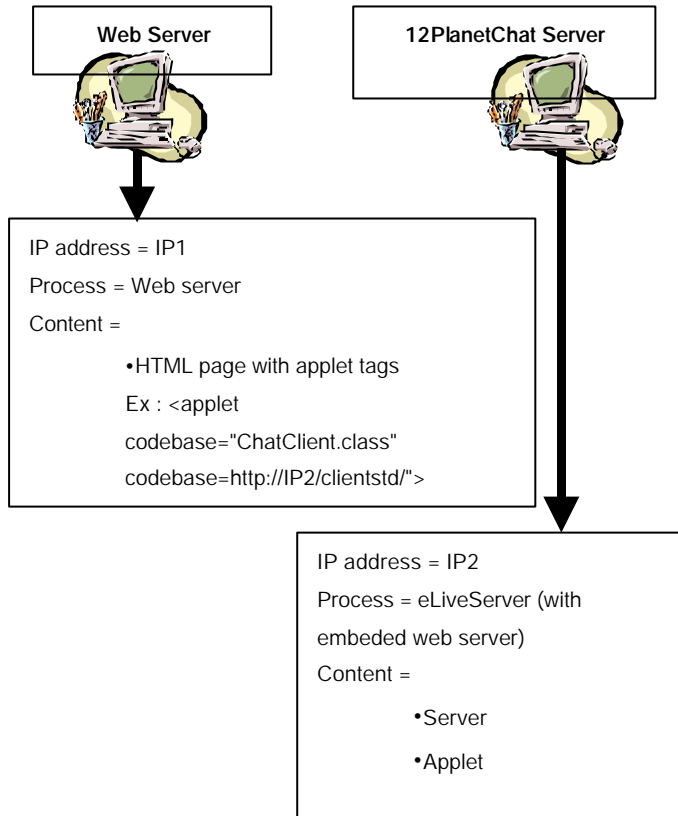
The Web Server software serves your "standard" web pages and chat web pages that contain dynamic interaction software that we call "Chat Client Applet".

There is no specific installation to do on the user side to deploy the chat applet, the software will be downloaded automatically on the user machine without any human intervention. This piece of dynamic software is very small (<50kb) so the installation is quite fast.

The eLiveServer™ software will accept connections from the chat client applet and make the necessary routing, processing work to complete the advanced interaction features.

2.2 Installation on a dedicated server

12Planet eLiveServer™ is installed on a different machine from your web server.



The Web Server software serves your "standard" web pages and chat web pages that contain dynamic interaction software that we call "Chat Client Applet".

The Chat Client Applet must be located on the dedicated server in order to communicate with it (due to Java security restrictions)

2.3 Login and integration with website

All login parameters for Chat applet can be given as parameters with the chat applet HTML page. This feature allows easy connectivity with custom user login system of your web site.

The eLiveServer™ can be configured to accept connections from only one particular chat page or a sub-domain, so users cannot bypass the login page verification before accessing the protected chat page(s). Once your own login control is done, you simply generate dynamically a chat page using a server side scripting tool (cgi, php, perl, java servlet, jsp, asp...) that will give the chat applet the appropriate login information as parameters. The chat applet will be set to connect directly to the chat server without asking users to enter their nickname again. This way the chat applet is used as a totally integrated communication component to your site.

3 eLiveServer Protocol

3.1 eLiveServer™ Server to Client protocol

Publicly useable protocol is the following. Only the protocol described below may be used or filtered. **All non described elements of the protocol should not be filtered except under explicit agreement with 12Planet.**

Server -> Client protocol	
nick <i>nickname</i>	Confirm the final username to the user
filter <i>shit;fuck;testword;</i>	Sends the list of filtered words
topic <i>publicgroup</i>	Sends the topic of the current room
join <i>+\$public\$publicgroup</i>	Sends the name of the current room
login <i>joe null</i>	Tells the user that joe just joined the room
channel <i>0-\$SYS\$users` 1+\$public\$publicgroup</i>	Sends the list of rooms to the user Nb of user <i>+\$group\$room`...</i>
list <i>joe@0</i>	Sends the list of users to the user username@waited time (optional, see server manual)
say <i>joe: hello joe</i>	Joe says joe: hello
pager <i>bob hello world +\$public\$publicgroup</i>	an instant message from bob from the room <i>+\$public\$publicgroup</i> saying hello
whisper <i>[[bob]]: how are you bob</i>	An private messge from bob saying <i>[[bob]]: how are you.</i>

The content of the message for the “say”, “pager” and “whisper” keywords support the live push sub-protocol using the @do\$ keyword, ex:

```
whisper|[[joe]]: @do$ /web http://www.12planet.com|joe
```

a push message containing the <http://www.12planet.com> hyperlink from joe.

3.2 eLiveServer™ Client to Server protocol

Client -> Server protocol

login joe <i>http://www.yahoo.com/chat.html</i> + <i>\$public</i> <i>\$publicgroup</i> <i>passwd</i>	Joe, from the page <i>http://www.yahoo.com/chat.html</i> asks to log into the server using <i>passwd</i> as the password, he intends to join the + <i>\$public</i> <i>\$publicgroup</i> chat room
channel + <i>\$public</i> <i>\$publicgroup</i> \$\$nopass <i>topic</i>	Join or create the + <i>\$public</i> <i>\$publicgroup</i> channel, with the system password \$\$nopass (meaning no password sent), setting the topic of the room to <i>topic</i>
say hello	Says hello to everyone
whisper whisper message bob	Send a private message to bob saying "whisper message"
pager paging message bob	Send an instant message to bob saying "paging message"

4 eLiveServer Bot API

4.1 *What is an eLiveServer Bot?*

An eLiveServer bot is an active component on the server side that is associated with a chat room and that performs a certain number of predefined actions at regular intervals such as getting information from an external source of information.

The bot has the possibility to send information to the chat room or any user within this chat room.

4.2 *Java Interface to create a bot*

```
package services;

public interface IService
{
    /** implement the doWork method to perform a task
     * during all the time the chat room is alive
     */
    public void doWork(infoserver.ChatRoom room);
}
```

4.3 *Sample Bot*

```
package services;

public class BasicService implements IService
{
    /** implement the doWork method so we perform the task
     * defined here during all the time the chat room is alive
     */
    public void doWork(infoserver.ChatRoom room) {

        /** get the information to send to the room*/
        String sMessage = "There are "+room.realsize()+" users in the room";
        room.sendToRoom("BasicService", sMessage);

        /** wait 5 seconds before working again*/
        pause(5000);
    }
}
```

```
/** help method for waiting some time*/
public void pause (long lDelay) {
    try {
        Thread.sleep(lDelay);
    } catch (Exception e) {
    }
}
}
```

4.4 Testing the plugin with eLiveServer™

Open the eliveserver.conf file, and add or modify bot name mappings, ex:

```
#Optional plugins
basicbot=services.BasicService
newsbot=services.NasdaqNews
```

Modify the grouplistserver.txt (or key.txt) configuration file in the 12planet_server/conf directory.

Define the inputplugin (filter client to server messages) and outputplugin (filter server to client messages) values, ex:

```
$public$testbot: bot=basicbot
```

If you need to create several bots, use the | sign to separate the bots, ex:

```
*$bot: bot=basicbot|newsbot
```

When the user joins the \$public\$testbot room (chat group public, chat room testbot), the bot will be activated and send a message every 5 seconds to the chat room.

5 eLiveServer™ plugin API

5.1 What is an eLiveServer™ plugin?

The eLiveServer™ offers a flexible server side development framework that allows customers to add their own message processing and handling mechanism on top of the standard eLiveServer™ core engine. A plugin defines 2 functions:

- one for processing incoming messages
- and one other for processing outgoing messages.

When the server receives a message, it will check whether if the room in which the user is in has one or more plugins defined (definition in the grouplistserver.txt file), if yes it will forward the message to the plugin. The plugin can then make the appropriate action (update a database, modify the message, send more messages to the user, reject the message...) and send the filtered message back to the server which continues the normal processing.

The same plugin will be called before the server sends a message to the user, and different type of processing may occur.

5.2 Java interface to create a plugin

The most basic plugin looks like the following:

```
package plugins;

public class MessageFilter implements IMessageFilter
{
    /**
     * Implement this class to create a filter that will filter incoming messages
     */
    public String filterIncomingMessage(String sOption, infoserver.Message m) {
        return m.getMsg();
    }

    /**
     * Implement this class to create a filter that will filter outgoing messages
     */
    public String filterOutgoingMessage(String sOption, infoserver.Message m) {
        return m.getMsg();
    }
}
```

The infoserver.Message class offer the getMsg() method to get the content of the message to be processed as a String.

The `infoserver.Message` class offer the `getUser()` method to get the user that is receiving or sending the message as an `UserProxy` object.

The `UserProxy` exposes the following APIs for scripting the user:

```
/** send a message to be processed by the server */
public void sendToServer(String dat) throws Exception;
/** send a message back to the user */
public void sendToUser(String dat) throws Exception;
/** get the complete name of the communication channel (chat room) in
which the user is in */
public String getFullChannelName();
/** get the web page url from which the user has logged into the
system */
public String getHomePage();
/** get the nickname of the user */
public String getAlias();
/** get the IP address of the user*/
public String getIP();
/** force the user to disconnect */
public void logout();
```

5.3 Testing the plugin with eLiveServer™

Open the `eliveserver.conf` file, and add or modify plugin name mappings, ex:

```
#Optional plugins
translationplug=plugins.Translator
std=plugins.MessageFilter
```

Modify the `grouplistserver.txt` configuration file in the `12planet_server/conf` directory.

Define the `inputplugin` (filter client to server messages) and `outputplugin` (filter server to client messages) values, ex:

```
*$translated: inputplugin=translationplug
```

If you need to create a piped line of plugins, use the `|` sign to separate the plugins, ex:

```
*$translated: inputplugin=translationplug|std
```

5.4 Optional translation plugin

Powered by  **WorldLingo**[™]
Translation Localization Globalization

An optional plugin for instant translation is available. Available languages are:

- Chinese

- English
- German
- Italian
- Japanese
- Korean
- Portuguese
- Russian
- Spanish

This plugin is a co-solution of 12Planet & WolrdLingo.

Please contact 12Planet for more information.

6 eLiveServer™ HTTP API

6.1 API calls

Server HTTP-API can be called **only by authorized IPs**, for security reasons. By default only "localhost" is authorized IP.

Authorized IP are defined in text file:

```
12planet_server/conf/trustedip.txt
```

If you are not authorized to use those APIs, you'll read "Unavailable/Unauthorized" from the server.

Programmers should use their own scripts (CGI, PHP, ASP, Servlet...) that make HTTP calls to the chat server and embed the result into their final HTML pages. If the scripting system is on the same machine as the chat server, then you'll simply need to add localhost and 127.0.0.1 into the trustedip.txt file.

6.2 API List

6.2.1 Introduction

All the APIs here are accessible from a web browser, actually the way they work is very close to the concept of a web service.

Each API may be called either under the format:

```
http://serverIP:httpport/apiname?param1=value1&param2=value2...
```

Or under the equivalent format:

```
http://serverIP:httpport/api?function=apiname&param1=value1&param2=value2...
```

You can either use those APIs directly from scripting tools and interpret the result or you can use them from our Java wrapping classes that will allow you to make direct function calls to Java wrappers. The Java wrappers use the eLiveWeb/RemoteAPI technology, you need to purchase a separate server developer licence in order to obtain it.

By appending the parameter xml=y, the result will be in XML format, ex:

```
http://server_address:10080/usercount?room=CHANNELNAME&xml=y
```

6.2.2 Number of people in one chat-room

Syntax:

```
http://server\_address:10080/usercount?room=CHANNELNAME
```

where CHANNELNAME is \$groupname\$roomname.

Ex: for the chat group infochat.com, and the chat room "the room", the API call would be:

```
http://localhost:10080/usercount?room=$portal.com$the+room
```

6.2.3 List of users in a chat-room

Syntax:

http://server_address:10080/userlist?room=CHANNELNAME

where CHANNELNAME is \$groupname\$roomname.

Ex: for the chat group portal.com, and the chat room "the room", the API call would be:

[http://localhost:10080/userlist?room=\\$portal.com\\$the+room](http://localhost:10080/userlist?room=$portal.com$the+room)

6.2.4 List of chat-rooms

Syntax:

http://server_address:10080/listallrooms?

6.2.5 Number of chat-rooms

Syntax:

http://server_address:10080/countallrooms?

6.2.6 Number of people in all chat-rooms

Syntax:

http://server_address:10080/countallusers?

6.2.7 Status of user: online/offline

Syntax:

http://server_address:10080/isonline?name=USERNAME

Ex:

<http://localhost:10080/isonline?name=john>

6.2.8 Send a message to a room

Syntax:

http://server_address:10080/sendtoroom?room=CHANNELNAME&message=MESSAGE

Ex:

[http://localhost:10080/sendtoroom?room=\\$public\\$a+room&message=hello+to+everyone](http://localhost:10080/sendtoroom?room=$public$a+room&message=hello+to+everyone)

6.2.9 Send a private message

Syntax:

http://server_address:10080/sendtouser?name=USERNAME&message=MESSAGE

Ex:

<http://localhost:10080/sendtouser?name=john&message=hello+to+everyone>

6.2.10 Stop server

Syntax:

http://server_address:10080/killserver?

6.2.11 General statistics about server

Syntax:

`http://server_address:10080/sysinfo?`

This API is provided for debugging purposes and is not currently documented and supported.

6.2.12 License information

Syntax:

`http://server_address:10080/licence?option=<property_name>`

List of available properties:

Property name	Value
MAXUSERS	Number
MAXROOMS	Number
MAXLANGUAGES	Number
HTTPTUNNELING	Activated / Not activated
ISMODERATED	Activated / Not activated
EXPIRATION	Date
MODULE	String (product name)

Other API are available for OEM projects, If you have any special needs, please, contact us

7 Fail over management

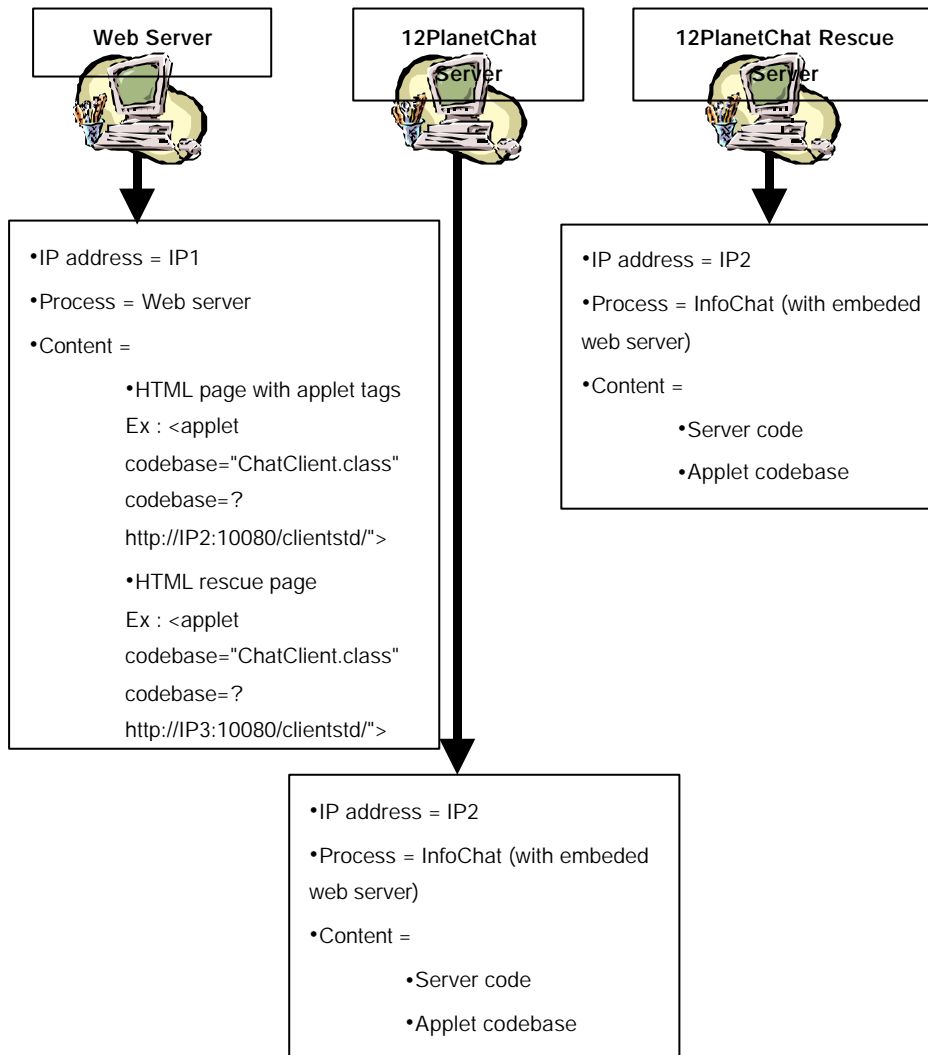
7.1 Chat Client fail over feature

From 12Planet Client side: parameter of chat client applet

```
<PARAM name = "sessionOnfailurl" value = "URL_name">.
```

When a user cannot connect to the server then the onfail URL is automatically opened. This URL can be used to establish a connection to a second server or to redirect toward a rescue page.

Typical rescue support configuration:



7.2 ELiveServer™

7.2.1 Process runner tool

To launch 12Planet Chat server in a secured process use the command:

```
java inforunner java infostart
```

If the process « java infostart » crashes, then a new process will be automatically rerun.

7.2.2 Monitoring

At any time once can test activeness of server by HTTP request:

http://server_name:10080/status?

Result is "12Planet Server is running" or a blank page. This message can be caught and integrated in a fail over application (Tivoli,...).

8 Web Server integration

The chat applet have to be stored on the same hosting machine as the eLiveServer™. For this reason there is always a need for a webserver on the hosting machine of eLiveServe.

12Planet eLiveServer™ contains an integrated webserver running by default on port 8080.

If you want to change the webserver you should copy (or create an alias) on directory:

```
12planet_server/www/
```

9 Proxy and firewall support

9.1 Introduction about HTTP tunneling

If your server is running on a public IP address, you can configure the server so users from behind corporate firewalls may access it.

12Planet ELiveServer use 2 connection mode to connect chat client with server:

- Socket mode (activated by default from port 7218)
- HTTPtunneling mode (automatically activated from port 10080 if socket mode has failed).

12Planet ELiveServer comes with an integrated http tunneling service which runs on the port 10080.

If you want to change this port, please follow the instructions depending on your configuration.

Configuration 12Planet ELiveServer is running on a dedicated machine

You should set the HTTP tunneling port of the chat server to the port 80 directly for improving performance. You should not use the HTTP connector servlet, but instead

Configuration 12Planet ELiveServer is running on the same machine as the web server.

You need either:

- a third party proxy service such as Apache proxy module to forward the requests from the port 80 to this port in order to make the http tunneling feature available on the port 80 from outside
- HTTPconnector servlet with any J2EE application server

9.2 Apache Proxy module

Install your Apache server with the proxy module included, ex:

Add ProxyPass lines to your http.conf file, ex :

```
./configure --enable-module=proxy;make ;make install

<VirtualHost 62.210.135.97>
ServerAdmin root@ mywebsite.com
DocumentRoot /home/chat/ a-directory /12planet_server/www
AccessFileName .htaccess
ServerName 62.210.135.97
CustomLog /usr/local/apache/logs/www. mywebsite.com/access_log
combined
ErrorLog /usr/local/apache/logs/www.planete-chat.com/error_log
    <Directory /home/a-directory/www.mywebsite.com>
        Order allow,deny
        Allow from all
        AllowOverride AuthConfig
    </Directory>
ProxyPass /servlet http://62.210.135.97:8080/servlet
ProxyPass /httpconnector http://62.210.135.97:10080/httpconnector
</VirtualHost>
```

This will map the /servlet and /httpconnector requests to the internal servlet engine on the port 8080 and the internal http tunneling engine on the port 10080

9.3 HTTPconnector Servlet

With HTTPconnector, 12Planet Chat Clients will always connect to httpconnector servlet in HTTP mode. The HTTPconnector will forward the HTTP messages from the standard port 80 of your web application server to the chat server internal HTTP tunneling processors located

HTTP connector servlet installation:

Step1

Copy to your web application server or your servlet environment all the files for the servlet httpconnector from directory:

```
12planet_server/httpconnector/
```

Step2

Define the environment parameter CHATSRV_HOST of the httpconnector, default value is <http://localhost:10080/>

This step is servlet engine specific)

1. Example with the SUN servlet reference implementation:

```
# <servletname>.code=<servletclass>
# <servletname>.initparams=<name=value>,<name=value>
httpconnector.code=httpconnector.httpconnector
httpconnector.initparams=CHATSRV_HOST=http://eLiveServerIP:10080/
```

2. Example with the tomcat servlet engine:

Add the <servlet> ... </servlet> portion to the web.xml configuration file.

```
<web-app>
```

```
...
```

```
    <servlet>
        <servlet-name>
            httpconnector
        </servlet-name>
        <servlet-class>
            httpconnector.httpconnector
        </servlet-class>
        <init-param>
            <param-name>CHATSRV_HOST</param-name>
            <param-value>http://localhost:10080/</param-value>
        </init-param>
    </servlet>
</web-app>
```

Test your http connector at <http://yourserver/servlet/httpconnector?>, you should see Infosuite ERR: java.lang.NullPointerException

Then on your chat page, add the following lines:

```
<param name="connectorHttpPort" value="80">  
<param name="connectorHttpPath" value="/servlet/httpconnector?">
```

Your chat applet is ready to work from behind firewalls.

10 Database integration

If the userdb option is set to “y”, and the database JDBC driver is correctly configured in the eliveserver.conf file and the JDBC driver classes are in the classpath of the Java VM that runs that chat server, then you'll be able to update the user online/offline information dynamically to a database and reuse this information from another application if necessary.

In the eliveserver.conf file, the following section should be configured:

```
#the SQL script file that should be used by the chat server, modify this file to use your own tables
```

```
dbscript=conf/sql/eLiveServer.sql
```

```
#Sample setup for the integrated hSql free JDBC database engine
```

```
jdbcdriver=org.hsql.jdbcDriver
```

```
dbuser=sa
```

```
dbpass=
```

```
dburl=jdbc:HypersonicSQL:databases/elivedb
```

```
dbvendor=hsql
```

```
#Sample setup for the mySql JDBC database engine
```

```
jdbcdriver=org.gjt.mm.mysql.Driver
```

```
dbuser=system
```

```
dbpass=
```

```
dburl=jdbc:mysql://localhost/infodb
```

```
dbvendor=mysql
```

```
#Sample setup for the Oracle JDBC database engine
```

```
jdbcdriver=oracle.jdbc.driver.OracleDriver
```

```
dbuser=system
```

```
dbpass=manager
```

```
dburl=jdbc:oracle:thin:@gunn:1521:planet
```

```
dbvendor=oracle
```

It's possible to modify the SQL scripts so you use your own tables and database selection/update scripts instead of the default ones.

The syntax of the SQL resource file is the following:

- all lines starting with the # sign are comments and are ignored
- dynamic data are defined by the chat server at the position of the {n} tags

Ex: dbSetOnline = update profilechat set online='{1}' where nick='{2}'

The dbSetOnline script (never modify the script name or it will stop working) will update the profilechat table and set the column online to the value of the parameter {1} given by the chat server and the column nick to the value of the parameter {2}.

11 Bandwidth for server

The used bandwidth by server depends on the way chat-rooms are used. Keys to estimate bandwidth are:

- Number of people per room: example, about 20 persons in one room
- Number of messages: example, about 1 message sent by one user every second (technical and user messages)
- Average size of messages: it is about 0.5kb

Server needs about $(20 \text{ sent messages by server per second}) * 0.5\text{kb}/1\text{sec} = 10\text{kb}/\text{sec} = 80\text{kbps}$ for optimum using.

12 Scalability and advanced set-up

12.1 *Number of concurrent users*

There is virtually no limit of concurrent users from 12Planet Server (there is still the limit of your licence. Please contact solution@12planet.com for upgrading). Limitation comes from OS and JVM. An optimised tuning of capacity of server is defined by following keys:

- Leap size (should be as big as supported by your configuration)
- File descriptor number (should be as big as supported by your configuration)
- Thread number (should be as big as supported by your configuration)

There is no general rules: these parameters depend on your OS and JVM choices.

12.2 *Quality of chat service*

The principle is to optimise thread management of the server.

The generated load on one server depends on the behaviour of users: the more users per rooms there are, and the more you need capacity on server (many small rooms require less resources than one very large room).

To improve speed you may:

- Increase the number of thread used by 12Planet Chat Server: number of used threads by 12Planet Server is defined in file: `eliveserver.conf`, we suggest to give the `nbsenders` and `nbreceivers` the number of concurrent connections you expect and to `nbprocessors` this value divided by 10. On some OS, such as Linux, you may need to rebuild the kernel in order to support a large number of simultaneous connections. Only the `nbprocessor` parameter affects the HTTP tunnelling clients.
- Install the `HTTPconnector` on more machines to split the load
- Increase RAM, if necessary
- Use a multiprocessor hosting machine
- Optimise the network connection (network card device / bandwidth...)

Generally speaking, we suggest to add servers if you want to offer a good chat-service for more than 2500 users.

Once you have reach the potential of your server you need to settle 2 or more machines and connect people with one or the other machine, depending on the room they request.

12.3 *Advanced tuning with Linux server Kernel 2.2.x*

To optimise the configuration with a Linux Server (for very high load)

1. Define optimised parameters

- Increase number of tasks
 - In file: `/usr/include/linux/tasks.h`
 - Parameter: `NR_TASKS`
 - Value: `512 -> 4090`
- Increase number of tasks per user
 - In file: `/usr/include/linux/tasks.h`
 - Parameter: `MAX_TASKS_PER_USER`
 - Value: `(NR_TASKS/2) -> NR_TASKS`
- Increase `NR_OPEN`
 - In file: `/usr/include/linux/limits.h`

- Parameter: NR_OPEN
Value: 1024 -> 4096
- Increase OPEN_MAX
In file: /usr/include/linux/limits.h
Parameter: OPEN_MAX
Value: 256 -> 4096
- Increase number of tasks per user
In file: /usr/include/linux/posix_types.h
Parameter: __FD_SETSIZE
Value: 1024 -> 4096
- Increase FD_SETSIZE
In file: In /usr/include/bits/type.h
Parameter: __FD_SETSIZE
Value: 1024 -> 4096
- Add
In file: /etc/security/limits.conf,
Add the lines:
 - * soft nofile 1024
 - * hard nofile 4096
- Add
In file: /etc/pam.d/login
Add:
 - session required /lib/security/pam_limits.so
- Increase system-wide file descriptor limit
In file: /etc/rc.d/rc.local
Add
 - echo 8192 > /proc/sys/fs/file-max
 - echo 24576 > /proc/sys/fs/inode-max

2. Recompile Kernel

- Step1.....
- Step2....

13 **ELiveServer™ and Instant Messaging**

ELiveServer™ can be installed as an Instant Messaging Server.

In standard this Messenger server is 100% compliant with Instant Messenger client from Microsoft™.

Instant Messenger client side can be customized to meet any custom needs. It is possible to develop corporate modules as well.

For community portal sites, it is possible to install a gateway to Yahoo network, MSN Network, ICQ Network and AOL Network.

Please contact 12Planet for information about these optional modules.