

FlashNow <http://www.nowcentral.com>

Documentation V1.4

Content:

- I. What is FlashNow**
- II. Installation**
 - 2.1 Windows**
 - 2.1.1 Install**
 - 2.1.2 Configure**
 - 2.1.3 Starting**
 - 2.1.4 Shutdown**
 - 2.2 Linux**
 - 2.2.1 Install**
 - 2.2.2 Configure**
 - 2.2.3 Starting**
 - 2.2.4 Shutdown**
 - 2.2.4 SysV startup script**
- III. Configuration**
 - 3.1 Command line options.**
 - 3.2 Configuration file format.**
 - 3.2.1 DTD section**
 - 3.2.2 Example Configuration**
 - 3.2.3 Configuration tags explained**
 - 3.3 Log format and options.**
 - 3.4 Remote administration**
 - 3.4.1 Getting the server configuration**
 - 3.4.2 Setting the server configuration**
 - 3.5 IP blocking**
 - 3.5.1 Example Configuration**
 - 3.5.2 Rules**
- IV. Registration**
- V. XML Syntax**
 - 5.1 Format**
 - 5.2 XML Version identifier**
 - 5.3 Structure / Nodes**
 - 5.4 Attributes**
 - 5.5 Requirements for Advanced Features**

VI. Working with FlashNow

6.1 The Basics

6.2 Simple Dynamic Flash Example "Echo"

6.3 Building a Basic Chat Room

6.4 Pushing Events

6.5 Server Extensions

6.6 Administrative Features

6.6.1 Disconnect

6.6.2 UserList

6.6.3 Channel List

VII. Other Resources (needs more URL's)

VIII. Tips and Tricks

Copyright 2001
Learning Worlds, Inc.

I. What is FlashNow

FlashNow enables Flash developers to create live real-time interactions over IP networks.

Applications include games, text and avatar chats, whiteboards, virtual classrooms, animated story environments, shared simulations, collaborative applications and other multiuser Flash-based facilities.

The **FlashNow** server works with the XML socket feature, opening a channel for messaging between different users. XML provides a structure for the message, but the server will support any alphanumeric string data.

The registered version of the **FlashNow** package runs under NT or Linux, and supports up to 64 simultaneous users. The enterprise version supports up to 1000 simultaneous users, but only runs under Linux. For testing and development purposes an unregistered version of **FlashNow** supports up to five users.

II. Installation

2.1 Windows Version

2.1.1 Install

Download the `FlashNowInstall.exe` from the **FlashNow** website (www.nowcentral.com). Save the file to your hard drive. Next, run the `FlashNowInstall.Exe` file and follow the installation instructions. The instructions will ask you to choose a directory in which to save the files.

2.1.2 Configure

see section **III. Configuration**

In most cases you can just start NowServer with the default configuration, and start up the examples. However, it is important that you read the Configuration section and properly configure your NowServer before putting it into a production environment.

2.1.3 Starting NowServer

Now run `NSBasic.exe` or choose from the start menu “Start | Programs | NowServer | Start NowServer” to activate the server. NowServer handles the XML socket messaging for all **FlashNow** applications. This must be active before running any of the example files.

2.1.4 Shutdown

In order to shut down the server run `NSBasic.exe -s` from a command prompt, or choose from the start menu “Start | Programs | NowServer | Shutdown NowServer”

In addition to the above shutdown method, when the server is run it creates a pid file (process ID). This file can be used by administration scripts such as the common *NIX “SystemV startup scripts” to find the pid of the server and shut it down.

NOTE: If you delete the pid file the server will nicely shut itself down.

2.2 Linux Version

2.2.1 Install

Unpack your distribution file.
(recommended install location `/usr/local/bin/NowServer`)

You will have downloaded a file named `flashNow.v1.4.tgz`. Untar this file with a command like:

```
tar xzvf flashNow.v1.4.tgz
```

This will create a NowServer directory. Change directories (`cd`) into this new directory. Change the permission on the NSBasic-1.4 or NSEnterprize-1.4 file to make it executable with a command like:

```
chmod 755 NSBasic-1.4
```

2.2.2 Configure

see section **III. Configuration**

With the Linux version you will probably need to read the Configuration instructions and edit the configuration file before running the NowServer. Common issues that you may encounter if you do not first edit your configuration are: Fail to start because the server can't read its configuration file, or can't write its pid file or log files.

2.2.3 Starting

Start server with the following command:

```
./NSBasic-1.4
```

Text similar to the following should appear:

```
20020117145957 : Enterprise Version
20020117145957  reading error messages file
default.err
20020117145957  reading messages file default.msg
20020117145957  Starting NowServer 1.4 Enterprise
Edition on port 5525
```

2.2.4 Shutdown

Stop the server with the following command:

```
./NSBasic-1.4 -s
```

In addition to the above shutdown method, when the server is run it creates a pid file (process ID). This file can be used by administration scripts such as the common *NIX "SystemV startup scripts" to find the pid of the server and shut it down.

NOTE: If you delete the pid file the server will nicely shut itself down.

2.2.4 SysV startup script

There is an example startup script included in the tar file "NowServerStartup". If you have extracted the NSBasic-1.4 binary to a location other than the suggested "/usr/local/bin/NowServer" you will need to edit this startup script to point to the location where you store the server binary.

NOTE: the default configuration uses the CWD as the storage location for the config and log files. If you start the server with this script the initial CWD will be `"/etc/rc.c/init.d"` so you will need to edit the configuration file to include the proper paths.

Also note that in the example `NowServerStartup` file we pass the server the `-c` flag with the full path to the configuration file for this same reason.

It is also recommended that you set the `pidfile` to `/var/run/`. This is where the shutdown script will normally look for it.

Now copy the `NowServerStartup` file to your `"/etc/rc.c/init.d"` directory and add it to your startup runlevels. (Use the `"ntsysv"` configuration tool to easily add the script to your runlevels.)

Using this script is optional. If you do not have root access or do not want to use the SysV mechanism, you will probably not be able to use this script.

III. Configuration

Upon starting, `NowServer` reads the **`NowServer.xml`**. If the file does not exist a default version will be created when you run `NowServer`. Edit this file with a basic text editor or XML editor to customize your server settings.

3.1 Command line options.

There are several command line parameters that can be passed to the server which can be used to override the values set in the configuration file. This is most useful when testing your server setup.

One important command line parameter is the `-c` flag which is used to pass a path and filename of the configuration file to use.

<code>-h</code>	<code>--help</code>	shows help screen
<code>-c configfile</code>	<code>--config=configfile</code>	sets the config path and file name (default value is <code>"NowServer.xml"</code> in the current working directory)

NOTE: If you are starting the server with a shell script such as a Unix style SysV startup script or if you are starting the

server as an NT service, you will want to pass the script the full path to the configuration file that you wish to use. (This is because the current working directory will be the location of the shell script in a Unix environment or the /windows/System32 directory in the case of an NT service.)

-v	--version	shows the program version
-p num	--port=num	sets the port number (overrides config file)
-r dir	--httprootdir=dir	sets the root directory for http server (overrides config file)
-s	--shutdown	shutdowns the server NOTE: if you should also pass the -c parameter with the shutdown, particularly if you are running multiple servers, the server will use the config file to determine the PID of the server you wish to shutdown.

3.2 Configuration file format.

3.2.1 DTD section

The DTD will assist XML editors in validating and maintaining acceptable values when editing your configuration file. If you edit the configuration with a text editor you can also refer to the DTD yourself to discover what the acceptable values can be.

```
<?xml version="1.0"?>
<!DOCTYPE NowServer [
<!ELEMENT NowServer (daemon, piddir, connection, log,
registration?, extensions, http, deniedIP?)>
<!ELEMENT daemon (#PCDATA)>
<!ATTLIST daemon
default (0|1) "0"
editable (yes|no) "no"
>
<!ELEMENT piddir (#PCDATA)>
<!ATTLIST piddir
default CDATA #IMPLIED
editable (yes|no) "no"
>
<!ELEMENT connection (transport, port, superchannel,
bindaddress)>
<!ELEMENT transport (#PCDATA)>
<!ATTLIST transport
default (TCP|any|HTTP) "any"
```

```
editable      (yes|no) "yes"
>
<!ELEMENT port (#PCDATA)>
<!ATTLIST port
default CDATA #IMPLIED
editable      (yes|no) "no"
>
<!ELEMENT superchannel (#PCDATA)>
<!ATTLIST superchannel
default CDATA #IMPLIED
editable      (yes|no) "no"
>
<!ELEMENT bindaddress (#PCDATA)>
<!ATTLIST bindaddress
default CDATA #IMPLIED
editable      (yes|no) "no"
>

<!ELEMENT extensions (enableuserlist, enablechannellist,
sendconf, remoteconf, adminpassword)>
<!ELEMENT enableuserlist (#PCDATA)>
<!ATTLIST enableuserlist
default (0|1) "1"
editable      (yes|no) "yes"
>
<!ELEMENT enablechannellist (#PCDATA)>
<!ATTLIST enablechannellist
default (0|1) "1"
editable      (yes|no) "yes"
>
<!ELEMENT sendconf (#PCDATA)>
<!ATTLIST sendconf
default (0|1) "0"
editable      (yes|no) "no"
>
<!ELEMENT remoteconf (#PCDATA)>
<!ATTLIST remoteconf
default (0|1) "0"
editable      (yes|no) "no"
>
<!ELEMENT adminpassword (#PCDATA)>
<!ATTLIST adminpassword
default CDATA #IMPLIED
editable      (yes|no) "no"
>

<!ELEMENT log (logmode, logdir, accesslog, errorlog,
priority, msgfile, errfile, addportnum, separatedhosts,
timestamp, time)> <!ELEMENT logmode (#PCDATA)>
<!ATTLIST logmode
default (none|file|screen|all) "file"
editable      (yes|no) "yes"
>
<!ELEMENT logdir (#PCDATA)>
<!ATTLIST logdir
default CDATA #IMPLIED
editable      (yes|no) "yes"
```

```
>
<!ELEMENT accesslog (#PCDATA)>
<!ATTLIST accesslog
default CDATA #IMPLIED
editable      (yes|no) "yes"
>
<!ELEMENT errorlog (#PCDATA)>
<!ATTLIST errorlog
default CDATA #IMPLIED
editable      (yes|no) "yes"
>
<!ELEMENT priority (#PCDATA)>
<!ATTLIST priority
default (1|2|3|4) "2"
editable      (yes|no) "yes"
>
<!ELEMENT msgfile (#PCDATA)>
<!ATTLIST msgfile
default (1|2|3|4) "2"
editable      (yes|no) "yes"
>
<!ELEMENT errfile (#PCDATA)>
<!ATTLIST errfile
default (1|2|3|4) "2"
editable      (yes|no) "yes"
>
<!ELEMENT addportnum (#PCDATA)>
<!ATTLIST addportnum
default (0|1) "1"
editable      (yes|no) "yes"
>
<!ELEMENT separatedhosts (#PCDATA)>
<!ATTLIST separatedhosts
default (0|1) "0"
editable      (yes|no) "yes"
>
<!ELEMENT timestamp (#PCDATA)>
<!ATTLIST timestamp
default CDATA #IMPLIED
editable      (yes|no) "yes"
>
<!ELEMENT time (#PCDATA)>
<!ATTLIST time
default (gmt|local) "local"
editable      (yes|no) "yes"
>
<!ELEMENT http (rootdir, poll, connections, nonkeepalive,
mime-type+)>
<!ELEMENT rootdir (#PCDATA)>
<!ATTLIST rootdir
default CDATA #IMPLIED
editable      (yes|no) "no"
>
<!ELEMENT poll (#PCDATA)>
<!ATTLIST poll
default CDATA #IMPLIED
editable      (yes|no) "no"
```

```
>
<!ELEMENT connections (#PCDATA)>
<!ATTLIST connections
default (direct|proxy|all) "all"
editable      (yes|no) "yes"
>
<!ELEMENT nonkeepalive (#PCDATA)>
<!ATTLIST nonkeepalive
default (0|1) "1"
editable      (yes|no) "yes"
>
<!ELEMENT WebServerLocation (#PCDATA)>
<!ATTLIST WebServerLocation
default CDATA #IMPLIED
editable      (yes|no) "yes"
>
<!ELEMENT mime-types (html, htm, class, jar, swf)>
<!ELEMENT html (#PCDATA)>
<!ATTLIST html
default CDATA #IMPLIED
editable      (yes|no) "no"
>
<!ELEMENT htm (#PCDATA)>
<!ATTLIST htm
default CDATA #IMPLIED
editable      (yes|no) "no"
>
<!ELEMENT class (#PCDATA)>
<!ATTLIST class
default CDATA #IMPLIED
editable      (yes|no) "no"
>
<!ELEMENT jar (#PCDATA)>
<!ATTLIST jar
default CDATA #IMPLIED
editable      (yes|no) "no"
>
<!ELEMENT swf (#PCDATA)>
<!ATTLIST swf
default CDATA #IMPLIED
editable      (yes|no) "no"
>

<!ELEMENT registration (username, serialnumber)>
<!ELEMENT username (#PCDATA)>
<!ATTLIST username
default CDATA #IMPLIED
editable      (yes|no) "no"
>
<!ELEMENT serialnumber (#PCDATA)>
<!ATTLIST serialnumber
default CDATA #IMPLIED
editable      (yes|no) "no"
>
<!ELEMENT deniedIP (deniedhost?, allowedhost?, deniedsubnet?,
allowedsubnet?)>
<!ELEMENT deniedhost (#PCDATA)>
```

```

<!ATTLIST deniedhost
default CDATA #IMPLIED
editable      (yes|no) "no"
>
<!ELEMENT allowedhost (#PCDATA)>
<!ATTLIST allowedhost
default CDATA #IMPLIED
editable      (yes|no) "no"
>
<!ELEMENT deniedsubnet (#PCDATA)>
<!ATTLIST deniedsubnet
default CDATA #IMPLIED
editable      (yes|no) "no"
>
<!ELEMENT allowedsubnet (#PCDATA)>
<!ATTLIST allowedsubnet
default CDATA #IMPLIED
editable      (yes|no) "no"
>

]>

```

3.2.2 Example Configuration

The following is an example configuration file.

```

<NowServer>
  <daemon          default="1"           editable="no" />
  <pidfile         default="."           editable="no" />
  <registration>
    <username       default="JohnDoe"     editable="no" />
    <serialnumber   default="55555555"    editable="no" />
  </registration>
  <extensions>
    <enableuserlist default="1"           editable="yes" />
    <enablechannellist default="1"       editable="yes" />
    <sendconf       default="0"           editable="no" />
    <remoteconf     default="0"           editable="no" />
    <adminpassword  default=""           editable="no" />
  </extensions>
  <log>
    <logmode        default="file"         editable="yes" />
    <logdir         default="logs"         editable="yes" />
    <accesslog      default="access.log"   editable="yes" />
    <errorlog       default="error.log"    editable="yes" />
    <priority       default="2"           editable="yes" />
    <msgfile        default="default.msg"   editable="yes" />
    <errfile        default="default.err"   editable="yes" />
    <addportnum     default="0"           editable="yes" />
    <separatedhosts default="0"           editable="yes" />
    <timestamp      default="YYYYMMDDhhmmss" editable="yes" />
    <time           default="local"       editable="yes" />
  </log>
  <connection>
    <transport      default="any"         editable="yes" />

```

```

        <port                default="5525"    editable="no"  />
        <superchannel       default="1"      editable="yes" />
        <bindaddress        default=""       editable="no"  />
    </connection>
    <http>
        <rootdir             default="./NowWebRoot"  editable="no"  />
        <connections        default="all"      editable="yes" />
        <nonkeepalive       default="1"       editable="yes" />
        <mime-types>
            <html           default="text/html"    editable="no"  />
            <htm            default="text/html"    editable="no"  />
            <class          default="application/octet-stream"
            editable="no" />
            <jar            default="application/java-archive"
            editable="no" />
            <swf            default="application/x-shockwave-
            flash"        editable="no" />
        </mime-types>
    </http>
    <deniedIP>
    </deniedIP>
</NowServer>

```

3.2.3 Configuration tags explained

<NowServer>	the main root tag
<daemon>	Specifies whether the server should start as daemon or console application (effects Linux version only) can be "0" or "1"; if "1", relay starts as daemon
<pidfile>	Sets the path where the PID file will be written. The server needs permission to write to this location. The PID file is used by administration scripts such as SysV startup scripts, SysV scripts will usually look for the pid value in /var/run/
<registration>	Parent node for registration data
<username>	Sets the username
<serialnumber>	Sets the serial number for registered versions
<extensions>	Parent node for several optional server features
<enableuserlist>	Should the server automatically return a userlist XML when a user joins or exits a channel. Enabling this feature will ensure that your userlist is up to date in almost all situations. You can disable this option and still have your clients call for a userlist update manually.

<code><enablechannellist></code>	Should the server return a channellist on request
<code><sendconf></code>	Should the server send its configuration to a client on request Recomended that you leave this value set to 0 except when you are initially setting up your server, or editing its configuration.
<code><remoteconf></code>	Should the server configuration be editable by clients via the remote administration features.
<code><adminpassword></code>	The administration password, required for use of adminstrative features, if there is no value set then any features that require this will not operate.
<code><log></code>	Parent node for log options
<code><logmode></code>	Output mode; can be "file", "screen", "all" or "none" File will print messages to the log file Screen will print messages on the console All will print to both the log file and the console None will disable logging.
<code><logdir></code>	Directory name for log files
<code><accesslog></code>	Sets the name of log file
<code><errorlog></code>	Sets the name of error log file
<code><priority></code>	Sets the priority of output of log messages If message has a priority value <= this log priority value it will output; otherwise it will not output.
<code><msgfile></code>	File that contains string patterns that will be written to the access log (these can be replaced with alternative strings for other languages). You can also customize the priority level of messages to shape your log file output.
<code><errfile></code>	File that contains string patterns that will be written to the error (these can be replaced with alternative strings for other languages) You can also customize the priority level of messages to shape your log file output.
<code><addportnum></code>	Should the server add port number to the end of the log filename. (useful on multi-server configurations)
<code><separatedhosts></code>	This feature will be coming soon

<code><timestamp></code>	Sets the timestamp pattern
<code><time></code>	Sets the time used to generate the timestamp to "GMT" or "local"
<code><connection></code>	Parent node for connection options
<code><transport></code>	<p>Sets the acceptable transport types for client connections.</p> <p>Should be "TCP", "HTTP" or "any"</p> <p>The flash5 XMLSocket is a TCP transport. The firewall client uses HTTP. Only the Enterprise version supports HTTP clients.</p>
<code><port></code>	<p>The port for client connections, the default is 5525.</p> <p>Flash5 XMLSocket connections only work on ports above 1024</p> <p>The HTTP flash client can use any port, but it is recommended that you use 80 for firewall tunneling for maximum effectiveness.</p>
<code><superchannel></code>	<p>The name of superchannel. The superchannel is a special feature. A client connected to this channel will receive messages from all channels and can send messages to any channel. See section 6.5 Working with FlashNow: Server Extensions. (Flash XML clients will need additional editing to work when connecting to the superchannel due to the need to include extra routing information to the messages it sends. The superchannel is useful for building "server plugins" such as archiving or administration systems, and is used for building relay chains of multiple servers.</p>
<code><bindaddress></code>	<p>IP addresses the server will listen on. Leave this value blank to have the server listen on all available IPs, or enter an IP address to restrict the server to only some IP addresses. (This is important if you want to use HTTP tunneling on port 80 and want to run the NowServer on the same computer as your Web server.)</p>
<code><http></code>	<p>Parent node HTTP options. The server has a very limited ability to act as a HTTPD server and send files to clients. This feature is not recommended but it is provided because it may be required for HTTP tunneling in some situations. See the firewall tunneling documentation for more information.</p>
<code><rootdir></code>	<p>The name of HTTPD server root directory. Any files you place in this directory will be accessible to the world.</p>

<code><connections></code>	Sets the type of client connection types that will be accepted as firewall tunneling client. Values can be "direct", "proxy" or "all".
<code><nonkeepalive></code>	Should the server support nonkeepalive connections. This may slow down the performance of the server, but most proxy servers will downgrade the flash connection type to a nonkeepalive connection.
<code><mime-types></code>	This section includes file extensions and mime-types for use by the httpd server functionality
<pre> <html> <htm> <class> <jar> <swf> </pre>	
<pre> <deniedIP> <deniedhost> <allowedhost> <deniedsubnet> <allowedsubnet> </pre>	optional tag; may include:

All tags, except parent tags, have two attributes:
"default" - this is the value used by the server at startup
"editable" - sets whether or not this attribute can be changed by remote administration; values can be "yes" or "no".

When the server reads config file, it stores these attributes in its internal structure as the default value and as the current value. Originally, the value of "current" equals the value of "default". However, when a client changes the server configuration by remote administration, it changes only the "current" attribute.

3.3 Log format and options.

The server usually keeps 2 log files: the first for common messages accesslog and the second for error messages (see `<accesslog>` and `<errorlog>` tags in the configuration).

One may change default settings to keep separate access log files for each client IP (`<separatedhosts default="1">`).

In which case the server will create a separate file for each unique client IP address.

If the `<timestamp>` tag exists in configuration file, the time is written at the beginning of each line of the log file separated from the message text by a tab character. The date format used is set in the configuration file, (for example, `<timestamp default="DD/MM/YYYY hh:mm:ss" />`)

The messages written to the log files can be customized by editing the `<msgfile>` and `<errfile>` (file names are set in the `<msgfile>` and `<errfile>` tags in the

configuration file).

These files contain lists of string patterns used to generate the log messages.

Here is an example of a pattern:

```
CL_BLOCK      1 <<client_IP>> rewrote config file
"<<filename>>"
```

The first field contains a message code (these are values set internally in the server).

The second - its priority, (this can be used to shape the type of data you want logged).

The third - the pattern itself. (you can change the text of the message you want printed to the log files).

In the above example, <<client_IP>> and <<filename>> will be replaced by the appropriate values provided by the server, and the whole message is then written to the access log when the server encounters the CL_BLOCK event.

3.4 Remote administration.

3.4.1 Getting the server configuration

The server may send its configuration to a client as a message in XML format when the following conditions are satisfied:

- 1) <sendconf > must be enabled in the configuration file and "default" attribute must be "1".
- 2) no other client has previously changed the "current" value of sendconf to disable it via remote administration.
- 3) "adminpassword" value sent with request is valid and matches the current value stored in servers configuration.

Here an example of client request and server answer:

Client - >Server:

```
<server getconf="" adminpassword="111" />
```

Server -> Client:

```
<NowEvents> <NowServer> <daemon current="0" editable="no" />
</piddir current="." editable="no" />
<registration> <username current="" editable="no" />
<serialnumber current="" editable="no" />
</registration> <extensions> <enableuserlist current="1"
editable="yes" /> <enablechannellist
current="1" editable="yes" /> <sendconf current="1"
editable="no" /> <remotefconf current="0" editable="no" />
<adminpassword current="111" editable="no" /> </extensions>
```

```

<log> <logmode current="all" editable="yes"
 /> <logdir current="logs/" editable="yes" /> <accesslog
current="access.log.5525" editable="yes" />
 <errorlog current="error.log.5525" editable="yes" />
<priority current="2" editable="yes" />
<msgfile current="default.msg" editable="yes" /> <errfile
current="default.err" editable="yes" />
<addportnum current="1" editable="yes" /> <separatedhosts
current="0" editable="yes" /> <timestamp
current="YYYYMMDDhhmmss" editable="yes" /> <time
current="local" editable="yes" /> /log> <connection>
<transport current="any" editable="yes" /> <port
current="5525" editable="no" /> <superchannel current="1"
editable="yes" /> <bindaddress current="" editable="no" />
</connection> <http>
<rootdir current="./NowWebRoot" editable="no" /> <poll
current="0" editable="no" />
<connections current="all" editable="yes" /> <nonkeepalive
current="1" editable="yes" />
<WebServerLocation current="" editable="yes" /><mime-
types><jar current="application/java-archive editable="no"
 />
<class current="application/octet-stream editable="no"
 /><swf current="application/x-shockwave-flash editable="no"
 />
<htm current="text/html editable="no" /><html
current="text/html editable="no" /> </mime-types> </http>
</NowServer></NowEvents>

```

3.4.2 Setting the server configuration.

The client may send commands to change the server configuration values by sending XML messages when the following conditions are satisfied:

- 1) Remote configuration must be enabled in the configuration file ("default" attribute must be "1").
- 2) changed parameter must be editable ("editable" attribute must be "yes").
- 3) no other client has previously changed the "current" value of remoteconf to disable it via remote administration.
- 4) "adminpassword" value sent with a request is valid and matches the current value stored in the server's configuration.

There are three <setconf> modes: <replace>, <reset>, and <replaceconf>

A <replace> message replaces the parameters setting the current value of that attribute to the value sent.

Example:

```

<server adminpassword="\111\"><setconf><replace
timestamp="\DD-MM-YYYY hh:mm:ss\"
time="\GMT\" /></setconf></server>

```

The Default value is still stored separately at the server to allow a reset. The <reset> messages will restore all the values to the default settings as they were set in the configuration file.

Example:

```
<server  
adminpassword=\"111\"><setconf><reset/></setconf></server>
```

The <replaceconf> message is identical to <replace> except it rewrites the configuration file with new parameters and saves the old configuration file with a .bak file extension instead of .xml.

3.5 IP blocking.

IP blocking can be used to block users who are abusing your server or rules, or to set up an intranet server that will only allow connections from local users.

3.5.1 Example Configuration

```
<deniedIP>  
<deniedhost default=\"127.0.0.1\" editable=\"no\" />  
<deniedhost default=\"127.0.0.2\" editable=\"no\" />  
<allowedhost default=\"127.0.0.3\" editable=\"no\" />  
<deniedsubnet default=\"127.0.0.1/8\" editable=\"no\" />  
<allowedsubnet default=\"127.0.0.1/24\" editable=\"no\" />  
</deniedIP>
```

3.5.2 Rules

When a client connects to the server the server will compare its IP address to the rules in the following order:

- i. if the IP is a deniedhost the connection is refused
- ii. if the IP is not included in a deniedsubnet it is accepted
- iii. if the IP is included as an allowedhost or included in an allowedsubnet it is accepted

Else the connection is refused.

Normally, for Internet use you would start with a very open configuration (the default is allow all), and add restrictive rules to block particular IPs or subnets. Alternatively for Intranet use you could start with a very restrictive rule such as <deniedsubnet default="???" editable="no" /> and then only allow particular clients or your local lan subnet.

IV. Registration

When you register **FlashNow** you will receive a username and registration number via E-mail. Edit the `NowServer.xml` with a text editor or XML editor. Add the registration node to the `NowServer` node.

```
<registration>
  <username          default="john doe"
  editable="no" />
  <serialnumber     default="984jc84jc8"
  editable="no" />
</registration>
```

When you restart `NowServer`, you will have full access to its features.

V. XML Syntax

XML is an approach to structuring data. You can use any valid XML message with `FlashNow` (in fact the server can relay almost any alphanumeric string).

However, if you want to make use of certain advanced `NowServer` functionality, such as Private Messaging, the Message Timestamp, Archiving Server plug-ins, or http tunneling you will need to use at least some of the syntax recommended below:

5.1 Format

```
<?xml version="1.0"?>
<NowEvents version="" encoding="" >
  <message timestamp="" channel="" user="" command=""
  type="" separator="" data="" targetuser="" target="" />
</NowEvents>
```

5.2 XML Version identifier

```
<?xml version="1.0"?>
```

This is not required by the server or flash, however some applications do require this string for valid parsing of an XML document.

5.3 Structure / Nodes

It is required that a valid XML document contain only one root level node. This node can contain additional nodes, and each of those can also contain still additional children. In the example, the root level node is `<NowEvents/>`. It would be valid to only send the `<message/>` node, however, if your application needed to send multiple messages they would have to be sent as separate messages or wrapped in a parent node. It is more efficient to send one longer message than multiple smaller messages.

In addition, when the message load on the server is high, or even when it is not, the server may receive 2 messages for the same client at virtually the

same time. In this case the Server would need to either buffer the messages and send them one at a time, or join them in a parent node. If the Server is under high load, and it were to buffer the incoming messages then it could fall farther and farther behind. Instead it will wrap multiple messages in a <NowEvents/> node.

If the messages are already wrapped in root level <NowEvents/> nodes it will strip them off and replace them with a single root level <NowEvents/> node. If the messages do not have a <NowEvents> node then it will simply wrap them as is into the new root level <NowEvents/> node.

It is therefore important that you account for the possibility of this additional level of XML hierarchy in your Flash actionscript code (or any other code which will be manipulating the XML messages of your application).

5.4 Attributes

version:	float, optional (reserved for future use)
encoding:	string (optional) usage: indicate usage of non-UTF message encoding
timestamp:	string (Do not generate this attribute on the client.) If you use this recommended format the NowServer can automatically add this parameter to your message node. This can be useful in complex interactions such as simulations and interactive gaming. This function can be disabled in the NowServer.xml configuration file. You can also set the format of the data and time in the configuration file. Any <message/> node will have a timestamp="" attribute added to it by the server if this feature is enabled. (enabling this feature is not required for HTTP tunneling however it is highly recommended).
channel:	string, (required for HTTP tunneling) usage: no effect when using XMLSocket but is required for use with "firewall tunneling" version of the NowServer
user:	string (required for HTTP tunneling) usage: useful for clients to track or display ID of user from which a message is received, and is required for use with "firewall tunneling" version of the NowServer
command:	string (optional) usage: indicates different actions to be taken on receiving this message.
type:	string (optional) usage: indicates type of "data" in a complex application where many message types are to be sent and need to be handled differently by your movie. possible values "string", "num", "list" (e.g. if num call parseInt on data, if list call split on data.

- separator:** char (optional) usage: only necessary if type is list (e.g. ", " for a csv list)
- data:** string (optional) usage: main body of the message (e.g. chat text, name of clip to play, list of x,y location coordinates)
- targetuser:** string (optional) usage: private messaging (only a client whose name matches this string will receive the message)

5.5 Requirements for Advanced Features

The <message/> node is not required for basic use, however, it is needed for the use of the timestamp, private messaging, and HTTP tunneling. It can be used with children or siblings of your own design and structure.

If you will only be using Standard FlashXML clients then you can omit the <message/> node.

If you wish to use the timestamp feature the only requirement is that a <message/> node be included in your message. It can be at any level in the XML hierarchy and can contain as many attributes or children as you wish.

If you wish to use the server's built in private messaging features then you must use the <message/> node with a targetuser="" attribute to direct the server in its message routing.

NOTE: the server can not currently route a message to multiple users via the private message feature.

If you intend to use the HTTP tunneling features of the Enterprise server then each message must contain the <message/> node as well as the channel="" and user="" attributes.

The additional attributes included in the suggested format above are not required, but are used in the example files available at www.nowcentral.com and are included here as an explanation and suggestion of how you can organize your data.

VI. Working with FlashNow

6.1 The Basics

FlashNow makes it possible to create Flash movies that communicate instantly with other Flash movies over the Internet. This means that you can use Flash to develop all kinds of synchronized multimedia and multiuser applications, from chats to games to virtual classrooms and collaborative environments.

To enable live interaction with **FlashNow**, your Flash movie must do three things:

- It must set up the XML socket and connect to the computer running the FlashNow server. This could be the localhost, or it could be a remote computer.
- The movie must be assigned a channel. This is a unique identifier (it can be a number or alphanumeric string) that allows you to establish multiple **FlashNOW** sessions on the same server.
- Finally, the movie must pass a name to the server to identify this instance of the client.

6.2 Simple Dynamic Flash Example (Echo)

`Echo fla` contains a simple Flash movie, with the bare minimum of functionality: it makes a connection with the NowServer, takes input, and then sends it back to the user.

Look at the first few lines of frame one:

```
channel = "echo";
userName = "Guest" + random(100);
server = "localhost";
port = 5525;
loadMovie ( "SocketFlashNow.swf", LiveCode );
```

This sets up the variable we use to connect to the server, then loads the swf `SocketFlashNow.swf`, into the empty MovieClip "LiveCode". You could put the code directly into the main file, but we load it from an external file so we can re-use it in other clients and easily make global changes.

The `SocketFlashNow.swf` clip contains the standard initialization routines for any `NowEnabled` movie. It creates a new XML socket and opens a connection to the server. It sends the `<SERVER CHANNEL="" NAME=""/>` message which is required by the server to assign the connection to a channel and route the message to the client.

Next, there is `newConnection()`, which is called as soon as the program connects to the server. It simply displays an appropriate message in `outputText`.

There is also an `init()` function called on successful connection, although there is nothing to initialize in this movie.

Finally the function `newXML()` is called by the server whenever any XML event is sent by any program on the same channel. It parses the XML data character by character, and sends it on to `parseMessage()`.

Back in the main swf file `Echo-1.3AutoSelect.swf`, there are two textfields, `inputText` and `outputText`. There is also a “send text” button, placed just offstage. The button is there to trigger events when receiving keystrokes, in this case the `<Enter>` key.

Back on the first frame of the main movie is the `parseMessage()` function. This function traverses the XML tree looking for a message node. The top-level XML node in this case is the `<NowEvents>` node so it moves on to the first child node. This first child node is a `<message>` node, so it looks to see if there is a command parameter with the value of “CHAT”. If there is, it places the value of the data parameter into `outputText`.

The “send text” button code should be fairly straightforward to understand. Whenever the `<enter>` key is pressed, it checks to make sure that there really is something in the `inputText` to send. If there is, it constructs an XML message containing the data, and sends it to the FlashNow server. Then it erases the `inputText` field.

This program contains all of the basic functionality of the FlashNow server.

6.3 Building a Basic Chat Room

From here it is fairly simple to expand `echo fla` into an actual chat room.

The most important new feature to add is the user list, and we label the messages with the name of the user who sent them. `basicChat fla` has a list of users, and automatically generates a random username.

Every time a new user joins, a user disconnects, or a connected user changes names a USERLIST message is automatically generated and sent to every movie on the channel. The XML sent by the server looks like this.

```
<?xml version="1.0"?>
<NowEvents>
  <message timestamp="20010524204646" user="NowServer"
    client="NowServer" channel="1"/>
  <USERLIST USERCOUNT="3">
    <USER name="Guest87"/>
    <USER name="Guest66"/>
    <USER name="Guest95"/>
  </USERLIST>
</NowEvents>
```

The **parseMessage()** function has some added checks to see if the message received was a USERLIST or USER.

- If the nodeName= USERLIST, it clears the user list in expectation of a new set of USER name values.
- If the nodeName=USER, it adds the value of the name parameter to the userList text variable.

For this example, users can change or input their name on the same page, using the “submit” button. The code on the submit button sets the local username variable to the text entered, and sends a **<SERVER NAME=" nameOfUser"/>** message to the server, just as it did after first connecting to the server.

***Note:** Instead of requiring the user to change their name on the same page, as in this example, you could construct a login page, with a little code (using PERL or PHP, for example) that passes the user name to Flash. You could also validate a registered user in the login process.*

In the example, you will also see a button called USERLIST. When clicked, this button sends another special SERVER XML command:

```
<SERVER GETUSERLLIST="" channel="theChannelOfThisMovie"
adminpassword="" />
```

(Adminpassword is optional)

Since a new user list is automatically sent each time a change occurs in the status of the connected users, this command may seem extraneous, and in fact when you click the button you will probably not see a change in the user list.

However you can turn off the server’s automatic user list feature (see the configuration section) and handle the user list by sending this message, using your own custom code, or you can eliminate the user list altogether. (Note that this method of getting a server-generated user list will work

even when automatic user list is turned off.)

Also notice that the name of the channel that the client joined is sent as an GETUSERLIST parameter. By using this function, you can request a user list for a foreign channel. However, to get a foreign list you must use the correct adminpassword. To get a list of all the channels open send <SERVER GETCHANNELLIST="" adminpassword="" /> You must use the correct adminpassword here as well to get a channel list returned to you.

This will return a message like

```
<?xml version="1.0"?>
<NowEvents>
  <message timestamp="20010524204646" user="NowServer"
    client="NowServer" channel="chatexample"/>
  <CHANNELLIST CHANNELCOUNT="2">
    <CHANNEL name="1"/>
    <CHANNEL name="chatexample"/>
  </CHANNELLIST>
</NowEvents>
```

In addition `basicChat.fla` adds one more script, attached to a symbol (a movie clip this time), that checks to see if the incoming chat text is still visible. When the user enters more lines of text than will fit in the `outputText` window, `scrollCheck` sets the `.scroll` property to `.maxscroll`, which keeps the last line of text in the text field visible.

6.4 Pushing Events

It is important to note that although most of the XML commands in the first two examples have been of the “CHAT” variety, there is no limit to the type of command that can be sent. This assumes that there is code somewhere to tell the client program what to do with each type of command.

`complexChat.fla` contains a similar chat program, except that it scans each input line for the characters `*LOL*`, the old IRC abbreviation for “Laugh out Loud”. If it sees these characters, it sends the “CHAT” command as normal, but it also sends a “PLAY” command. This message uses the data parameter to pass a string that is the name of a sound object declared in `init()`.

The “SEND TEXT” button that scans for the <enter> key has a bit more complex code this time. It starts out as before, checking that there really is data in the `inputText` field.

Then it starts an XML message, but does not include the `</NowEvents>` tag to close it. It converts the input string to upper case, and then searches it for `*LOL*`. If the text is found (note that the `indexOf()` method returns -1 if it is not found), then a second message tag is added.

This is what tells Flash to play the “laugh” sound.

This example also uses the **escape()** function to deal with characters such as the double-quote. The **parseMessage()** function is modified to **unescape()** the data that it receives. Try typing “*I can’t live without FlashNOW*” in `basicChat.fla`, and then in `complexChat.fla`.

In this release of FlashNow, we have added a Private Message feature. First note that when we build the user list, the `userList` dynamic text field now has the HTML checkbox marked. In **parseMessage()** we build a HTML document, with each name a href link.

The strange format of the URL in the href tag ...

```
<A HREF="asfunction:_root.PrivateMessage,Guest99">
```

...makes use of an undocumented Flash feature, **asfunction:**. The “asfunction:” feature works much like the “javascript:” URL syntax, but rather than calling a javascript function included in the movie’s parent HTML page, the “asfunction:” URL calls an ActionScript defined inside the Flash movie itself. In this case we call the **PrivateMessage()** function and pass it the name of the user we just clicked on.

The **PrivateMessage()** function is also included on frame 1 of our Flash movie. Examining this function, we see that it looks almost exactly like the ActionScript used on the “send text” button. The important difference is the addition of a `targetuser` parameter in the message being sent:

- When using the suggested **<MESSAGE />** syntax the server will look for a `targetuser` in the sent XML.
- If none is included, then the message is sent to all clients on the channel.
- If there is a `targetuser`, the message is sent only to those clients whose names match the value sent for `targetuser`.

There are many other useful commands that could be implemented in addition to those just described (“**SERVER NAME**”, “**SERVER CHANNEL**”, “**SERVER GETUSERLIST**”, “**CHAT**” and “**PLAY**”). Anything that Flash 5 can do can be communicated from one Flash application to another through an XML message.

This example also includes basic scrollbars to view previous lines of text. There are four new variables for each scrollbar: **percent**, **start**, **location**, and **max**.

- **Percent** keeps track of the percentage of the scrollbar that the handle has covered.
- **Start** is the pixel coordinate of the top of the scrollbar (where the

handle starts).

- **Max** is the bottom of the bar (where the handle ends).
- **Location** is the current location of the text being scrolled.

Thirty times a second - **onClipEvent(enterFrame)** - **location** is computed, based on the text's **.maxscroll**, and **.scroll** is set.

6.5 Server Extensions

superchannel

Included in the examples are some Perl clients. These run as a client to the NowServer, and can be used for testing as well as to develop additional server functionality. Additional Documentation for these clients are included with those examples.

6.6 Administrative Features

The NowServer v1.4 has a few new administration features. The Admin-1.4.fla shows how these can be used. Some of the administration features require the adminpassword to be set in the NowServer.xml configuration document, and must pass a matching adminpassword with the message.

6.6.1 Disconnect

A user can request to disconnect themselves.

```
<SERVER DISCONNECT="" CHANNEL="" NAME="" />
```

With the adminpassword a user can request to disconnect anyone.

Channel indicates on what channel to look for the user and targetuser indicates the username to disconnect

```
<SERVER DISCONNECT="" CHANNEL="" targetuser=""  
adminpassword="" />
```

6.6.2 UserList

A user can request an update to the User List. The updated userlist is sent to all users on that channel

```
<SERVER GETUSERLIST="<mode>" CHANNEL="<name>" />
```

The server returns an XML document containing the names of users connected to the channel specified as <name>; if <mode> equals "full" the server returns additional information about each user: IP address, connection time and last message time. If the CHANNEL value is not included then the value is assumed to be that of the current channel the client is connected to.

With the adminpassword a user can request a User List for a different channel than the one they are connected to.

```
<SERVER GETUSERLIST="<mode>" CHANNEL="<name>"
adminpassword="" />
```

In the Enterprise edition, the server keeps two userlists for local and remote users (connected to another servers which are lined by the relay server).

The enhanced version of <getuserlist> request gives the ability to get these lists separately:

```
<SERVER GETUSERLIST="<mode>" TYPE="<type>"
CHANNEL="<name>" />
```

where <type> can be "local", "remote" or "all".

Example I.

Client ->Server: (Client sends this message to the Server)

```
<server getuserlist="" channel="dsdsf" />
```

Server -> Client: (Server responds to Client with this message Client:)

```
<?xml version="1.0"?><NowEvents><message
timestamp="09-01-2002 16:12:03" user="NowServer"
client="NowServer" channel="dsdsf"/><USERLIST
USERCOUNT="1"><USER
name="dsfsd"/></USERLIST></NowEvents>
```

Example II.

Client -> Server:

```
<server getuserlist="full" channel="dsdsf" />
```

Server -> Client:

```
<?xml version="1.0"?><NowEvents><message
timestamp="09-01-2002 16:12:28" user="NowServer"
client="NowServer" channel="dsdsf"/><USERLIST
USERCOUNT="1"><USER name="dsfsd" IP="127.0.0.1"
ConnectTime="09-01-2002 16:09:14" LastMessageTime="09-
01-2002 16:12:28"/></USERLIST></NowEvents>
```

6.6.3 Channel List

The adminpassword is required to get a Channel List

```
<SERVER GETCHANNELLIST=" <mode> "  
ADMINPASSWORD=" <passwd> " />
```

Mode can be:

“brief” returns a list of channels only
“medium” returns a multi level list, channels and usernames. Each CHANNEL node contains the userlist for that channel
“full” returns a multi level list, channel and usernames. In this case the userlist contains additional information on each user such as their IPs, Connection times and last message times.

All messages in the format <SERVER /> are not relayed to the other clients on that channel, including those above and the <SERVER CHANNEL="" NAME=""/> message you use when first connecting to the server or changing your identification.

Example I.

Client ->Server: (Client sends this message to the Server)

```
<server getchannellist="brief" adminpassword="111" />
```

Server -> Client: (Server responds to Client with this message Client:)

```
<?xml version="1.0"?><NowEvents><message  
timestamp="21-01-2002 13:06:11" user="NowServer"  
client="NowServer" channel="5"/><CHANNELLIST  
CHANNELCOUNT="6"><CHANNEL name="555" /><CHANNEL  
name="1" /><CHANNEL name="2" /><CHANNEL name="-1"  
/><CHANNEL name="testing" /><CHANNEL name="5"  
/></CHANNELLIST></NowEvents>
```

Example II.

Client ->Server: (Client sends this message to the Server)

```
<server getchannellist="medium" adminpassword="111"  
/>
```

Server -> Client: (Server responds to Client with this message Client:)

```
<?xml version="1.0"?><NowEvents><message  
timestamp="21-01-2002 13:10:52" user="NowServer"
```

```

client="NowServer" channel="5"/><CHANNELLIST
CHANNELCOUNT="6"><CHANNEL name="555"><USERLIST
USERCOUNT="1"><USER
name="Ghost" /></USERLIST></CHANNEL><CHANNEL
name="1"><USERLIST
USERCOUNT="0"></USERLIST></CHANNEL><CHANNEL
name="2"><USERLIST USERCOUNT="2"><USER
name="Guest1" /><USER
name="Guest2" /></USERLIST></CHANNEL><CHANNEL name="-
1"><USERLIST
USERCOUNT="0"></USERLIST></CHANNEL><CHANNEL
name="testing"><USERLIST USERCOUNT="1"><USER
name="Guest33" /></USERLIST></CHANNEL><CHANNEL
name="5"><USERLIST USERCOUNT="1"><USER
name="Guest3" /></USERLIST></CHANNEL></CHANNELLIST></N
owEvents>

```

Example III.

Client ->Server: (Client sends this message to the Server)

```
<server getchannellist="full" adminpassword="111" />
```

Server -> Client: (Server responds to Client with this message Client:)

```

<?xml version="1.0"?><NowEvents><message
timestamp="21-01-2002 13:15:11" user="NowServer"
client="NowServer" channel="5"/><CHANNELLIST
CHANNELCOUNT="6"><CHANNEL name="555"><USERLIST
USERCOUNT="1"><USER name="Ghost" IP="127.0.0.1"
ConnectTime="21-01-2002 13:01:54" LastMessageTime="21-
01-2002 13:02:15"/></USERLIST></CHANNEL><CHANNEL
name="1"><USERLIST
USERCOUNT="0"></USERLIST></CHANNEL><CHANNEL
name="2"><USERLIST USERCOUNT="2"><USER name="Guest1"
IP="127.0.0.1" ConnectTime="21-01-2002 12:59:57"
LastMessageTime="21-01-2002 13:00:34"/><USER
name="Guest2" IP="127.0.0.1" ConnectTime="21-01-2002
13:00:36" LastMessageTime="21-01-2002
13:00:58"/></USERLIST></CHANNEL><CHANNEL name="-
1"><USERLIST
USERCOUNT="0"></USERLIST></CHANNEL><CHANNEL
name="testing"><USERLIST USERCOUNT="1"><USER
name="Guest33" IP="127.0.0.1" ConnectTime="21-01-2002
13:01:25" LastMessageTime="21-01-2002
13:01:49"/></USERLIST></CHANNEL><CHANNEL
name="5"><USERLIST USERCOUNT="1"><USER name="Guest3"
IP="127.0.0.1" ConnectTime="21-01-2002 13:01:03"
LastMessageTime="21-01-2002
13:15:11"/></USERLIST></CHANNEL></CHANNELLIST></NowEv
ents>

```

VII. Other Resources

Macromedia Flash 5 documentation:

ActionScript Reference Guide First Edition, July 2000

- Chapter 5: Integrating Flash with Web Applications (p125-136)
- Chapter 7: Action Script Dictionary, XML Object (p399-426)

VIII. Tips and Tricks

Flash Security Model

It is important to note that when you are ready to post your NowEnabled applications, the NowServer will need to be running on the same domain as your web server or a sub domain of the web server. In the simplest case you will simply run the NowServer on the same computer as your web server. However, if you have multiple web servers you should look at the Flash documentation regarding the Flash security model.

The only exception to this is that a `.swf` file loaded from a local drive can connect to any server, so you can connect to a remote NowServer easily when authoring your Flash application.

Refer to the examples found at NowCentral.com for specific examples.
http://www.nowcentral.com/demo/flashNOW/SecurityTest/Test_ChatWN.html
<http://www.nowcentral.com/demo/flashNOW/SecurityTest/securityanimation.html>

It is because of these restrictions that you may need to use the NowServer's Web serving function in order to use HTTP tunneling. If you have multiple sub domain names such as `www.nowcentral.com` and `flash.nowcentral.com` you can run your main Web server on `www.nowcentral.com:80` and the tunneling NowServer on `flash.nowcentral.com:80` and files loaded from `www.nowcentral.com:80` will have permission to connect to the server at `flash.nowcentral.com:80`. However, if you do not have multiple sub domains of the same name you will want both a Web server and the NowServer running on port 80 on the same host. This can be done with third party configurations, or as mentioned before the NowServer has rudimentary Web server functionality.

Channels

You can have multiple channels running on one NowServer. Each NowServer client has its own message buffer, and in addition each channel has message routing, and userlist completely separate from other channels. All users on a channel will receive messages from all other users on the same channel. You can use almost any string as a channel identifier, however certain punctuation characters can cause problems so it is recommended that you URI escape values sent to the server if they originated as user input.

If you look in the **NowServer.xml** configuration file you will note a parameter called `superchannel`. By default this is set to 1. This

superchannel is used by server plugins, and to synchronize multiple servers into a server farm. Therefore, any normal Flash client which attempts to connect on this channel will not get messages in the form expected, and messages sent from it will not be properly routed. It is possible to write flash clients that operate on the superchannel by adding additional XML parsing and routing information to the XML messages. The server examples provided work with normal channels and the superchannel.

The superchannel will receive all messages sent by any user connected to the server regardless of their channel. Additionally a client on the superchannel must specify a target channel when sending messages, and can therefore send messages to any channel. Using these features you can build synchronized applications that support tens of thousands of simultaneous users, or systems and games that require server side logic like Agents, bots, archivers, or game rule managers.

Check back at nowcentral.com for more documentation and example code for building server plugins and chat bots. The PERL scripts included in the install show examples of how to use the Superchannel.

Latency

Latency is the delay between when you send a message and when everyone else receives it. Latency with the NowServer is extremely low. Normally it is below 300ms, whereas a HTTP chat will often have a latency of over a second. However, network conditions (e.g. the Internet "weather") will affect your latency.

Applications like Chat and turn-based games should not have any problems with the latency issue in all but the worst conditions.

However interactive games where all users are playing at the same time will be sensitive to the message latency. There are many ways of dealing with this latency. One is to design your game so that the speed of play is slow enough to absorb the differences in time.

When doing this you can test latency by counting how long it takes for a client to get its own messages back, and warn users when they fall below an acceptable latency. Also remember that latency on your local network will often be less than that experienced by far flung internet users.

Another option is to use the Timestamp parameter, which the NowServer can add to your message, to mediate which action takes precedence.

Note: that if you are using HTTP tunneling as an option for users behind a firewall their latency will be very high compared to normal XMLSocket connections (you should expect latency as high as 2 seconds or more).

Message Frequency

While normally not recommended, it is possible and in some

circumstances viable to send messages more frequently than the message latency. The Server will buffer multiple incoming messages and wrap them into a single XML message if more than one message is received for a single client before there is an opportunity to send it to the client. This can happen if your clients send messages more quickly than the latency or even in the infrequent situation when two clients send messages simultaneously. If you are using http tunneling you will almost certainly have this effect since the server must buffer input until the client requests an update and the minimum polling time we have found effective is 1 second.

If you are using the recommended XML format of <NowEvents> and <message> all the <message> nodes will be wrapped inside one <NowEvents> node.

If you are using your own nodes the complete messages will be wrapped inside a <NowEvents> node. This needs to be done because if the messages are simply strung one after the other without a common parent node they will break the expected XML syntax rules and the client will only parse the first node.

If the server were to delay relaying the messages and try to serialize them one at a time a backlog can form and the messages can get more and more out of sync if the volume or frequency of messages is higher than the message latency. It is therefore necessary that when you are writing your code to handle the XML messages that you take this possibility of multiple messages into account.

Also important to note is that during our stress testing of the server under high load, the flash clients start to degrade before our JAVA or PERL clients under high message load. In other words when the message frequency being sent to a client is high the clients seems to start actually missing messages. Flash XMLSocket seems to be susceptible to this problem particularly on slower computers.

SimpleNowArchiver

The SimpleNowArchiver is a sample client written in PERL, which connects to the NowServer and prints all messages received to the console. This tool can be very useful while developing and debugging your FlashNow applications. You may pass the SimpleNowArchiver several command line parameters, listed below with the default values

```
port -p 5525
channel -c 1
name -n Archiver
host -h localhost
```

Example usage:

```
SimpleNowArchiver.exe -h yourserver.com -p 1024
```

- The SimpleNowArchiver supports the “superchannel” (channel 1) mentioned earlier in configuration, which means that if you use the default channel value “1” the SimpleNowArchiver will print messages sent to all the channels in use on you NowServer.
- To exit the SimpleNowArchiver, type Ctrl-C in the console.

An .exe and the PERL Source for SimpleNowArchiver are included with the development tools. If you have PERL installed you can use and modify the SimpleNowArchiver.pl file to add server side functionality to your system.

BeerClient

The BeerClient is another sample client written in PERL, which can be used for debugging. The BeerClient sends messages to the NowServer automatically on a timed basis. It has the same command switches as SimpleNowArchiver, as well as a few additional command line parameters, listed below with the default values:

```
port -p 5525
channel -c 3
name -n BeerClient
host -h localhost
command -k CHAT
message -m $i on the wall
delay -d 3
```

Example usage:

```
BeerClient.exe -h yourserver.com -p 1024 -k DRINK -m
"$i bottles to go"
```

- **Command** and **message** allow you to customize the XML message sent to a small degree. If you use “\$i” in your message string, it will be replaced with a countdown value starting at 99 and decreasing by one for each message until its value is 1.
- **Delay** is the number of seconds BeerClient waits between sending XML messages.
- The BeerClient will send 99 messages and then automatically exit. (Hence the name. “99 bottles of beer on a wall, 99 bottles of beer...”)
- To exit the BeerClient manually type Ctrl-C in the console.

An .exe and the PERL Source for BeerClient are included with the development tools. If you have PERL installed you can use and modify the BeerClient.pl file to add server side functionality to your system.

HTTP messaging / Quick Messaging

Besides its use in firewall tunneling, you can make use of the NowServer's http transport for sending messages from web links. You could use this to build external controls for your system in html.

To send a message from HTML, you will have to use the recommended XML syntax in your flash movie (see above). To send the message construct a HTML link like this:

```
<A HREF="http://myserver.com:5525/cgi-bin/message?n1=<hello world/>">hello world</A>
```

This is NOT really a CGI script, nor is there any way to run external scripts through the NowServer. The URL string is simply formatted in a familiar way, because the server needs to look for something to know how to handle the request (and look like proper http requests to a firewall). This should make it easy to make links or forms that send messages to your server.

Escape User Input

Whenever you are sending strings of unknown content, such as the chat messages used in the examples, it is recommended that you escape() the string before sending it to the NowServer. This is necessary because, if the string contains any characters that will break the XML syntax, your message will be ignored or may cause an error in your Flash file.

For instance, if the message looks like:

```
<message data="hi bob"/>
```

... but the user types a quoted word in the message, then the message sent is actually something like this:

```
<message data="bob you are "so" cool"/>
```

Effectively the word **so** is now unquoted and the whole message is no longer valid XML. Now if we first escape the message, with the following code:

```
TextInput = "bob you are \"so\" cool";  
// notice the need to escape the quotes here.  
CleanInput = escape(TextInput);  
XMLMessage = '<message data="' add CleanInput add \  
"/>';  
// in this string assignment we use single quotes to
```

```
// encapsulate the strings to avoid having to escape
// the double quotes
```

The ActionScript `escape()` function is designed to prepare values to be sent in a URL CGI request, so the text that is sent in this case is:

```
<message data="bob%20you%20are%20%22so%22%20cool"/>
```

`escape()` changes the spaces to `%20` and the double quote “ to `%22`, we didn't have any problem with the spaces but changing them to `%20` doesn't cause any problems either. It does however make the message unreadable so we need to call `unescape()` on it when we receive it back from the XML Socket connection.

```
XMLData = "bob%20you%20are%20%22so%22%20cool";
OutputText = unescape(XMLData);
```

Now we have back our original text ready to display on screen.

See Macromedia's documentation at:

http://www.macromedia.com/support/flash/ts/documents/url_encoding.htm
http://www.macromedia.com/support/flash/action_scripts/functions/escape.html
http://www.macromedia.com/support/flash/action_scripts/functions/unescape.html

asfunction

In some of the examples we make use of an undocumented flash feature called `asfunction`.

This allows a `<A href>` link in a flash html text field to call a function defined in the movies actionscript.

For more information on this function, check:

http://www.actionscripts.org/tutorials/intermediate/Invoking_Actions_From_HTML_Textfields/index.shtml

If you are not already familiar with functions, you might want to review the following:

http://www.macromedia.com/support/flash/action_scripts/actions/function.html

Function target paths

The flash player 5,0,30,0 which is still very common, and is the version shipped with the flash CD has a bug which manifests as a difference in

the context from which functions are called in the standalone player and in a browser.

If you call a second function from within a function that was triggered by the XMLSocket.onXML or XML.onLoad events, it calls from the context of _root in the stand alone player and authoring environment. But when used in IE ActiveX or Netscape plug-in, it calls correctly from the context of the movie clip.

In our examples we assign:

```
NowServer.onXML = newXML;
```

Then in newXML we call:

```
_root.parseMessage(DataRecieved.firstChild, 0);
```

Since this is in a movie clip one level down from the _root where **parseMessage()** is defined, it would appear that we could call `_parent.parseMessage()`, but due to the inconsistency in the context from which newXML is triggered by the XMLSocket object, this would fail in the standalone player.

There are a couple of different solutions to avoid this problem. One solution is to always use full paths starting from _root. Another solution is to create an alias to the target movie on the _root or _level0 (this is also useful when using multiple levels in different flash movies as well).

Reserved Messages

The special **<SERVER />** message format is used for server directives. Messages in this format will not be relayed to other clients.

In addition there is also a **<POLL />** message format used by the HTTP Tunneling clients. Any messages that appear to be poll requests will not be relayed to other clients.

Additionally if you are intend to use your own XML format, and not use the special features of the NowServer we recommend that you avoid having any of your nodes appear similar to the recommended `<NowEvent>` or `<message>` nodes explained above to avoid unexpected conflicts and to avoid having the server unnecessarily try to parse your messages.

Flash -- XML Parsing

In the examples we use a XML parsing method which will read through every node in an XML tree regardless of it's length or depth, but it does not track what the parent or level of each node is. For the short

messages likely to be used with realtime messages this is usually quite sufficient. If you are using a more complex XML structure and attempting to call directly to a particular node you may encounter problems when using NowServer since at times the Server will wrap multiple client messages into a new <NowEvents> root level node. The reasons that these new root level nodes are needed have been covered previously in this documentation. One solution for you may be including the <NowEvents> node in all your messages since the server will not add another level in that case merely join multiple messages, in one common <NowEvents> node. However as the examples show, what we recommend is to use a recursive function like the one used in the examples.

FlashNow, NowServer, and NowEnabled are registered trademarks of Learning Worlds, Inc.

FlashNow © 2001, Learning Worlds, Inc. All rights reserved