

# **ADOBE® FLASH® MEDIA SERVER**

## **ADMINISTRATION API REFERENCE**

© 2007 Adobe Systems Incorporated. All rights reserved.

Adobe® Flash® Media Server Administration API Reference

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, and Flash are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Microsoft, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are the property of their respective owners.

Portions include software under the following terms:

**Sorenson  
Spark.** Sorenson™ Spark™ video compression and decompression technology licensed from Sorenson Media, Inc.

Licensee shall not use the MP3 compressed audio within the Software for real time broadcasting (terrestrial, satellite, cable or other media), or broadcasting via Internet or other networks, such as but not limited to intranets, etc., or in pay-audio or audio on demand applications to any non-PC device (i.e., mobile phones or set-top boxes). Licensee acknowledges that use of the Software for non-PC devices, as described herein, may require the payment of licensing royalties or other amounts to third parties who may hold intellectual property rights related to the MP3 technology and that Adobe has not paid any royalties or other amounts on account of third party intellectual property rights for such use. If Licensee requires an MP3 decoder for such non-PC use, Licensee is responsible for obtaining the necessary MP3 technology license.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Notice to U.S. Government End Users. The Software and Documentation are “Commercial Items,” as that term is defined at 48 C.F.R. §2.101, consisting of “Commercial Computer Software” and “Commercial Computer Software Documentation,” as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

# Contents

<b>Administration API Reference</b>	1
addAdmin()	1
addApp()	2
addVHostAlias()	3
approveDebugSession()	4
changePswd()	5
gc()	6
getActiveInstances()	7
getActiveVHosts()	8
getActiveVHostStats()	9
getAdaptors()	11
getAdminContext()	12
getAdmins()	13
getApps()	15
getAppStats()	16
getConfig()	18
getConfig2()	20
getFileCacheStats()	23
getGroupMembers()	25
getGroupStats()	26
getGroups()	28
getInstanceStats()	29
getIOStats()	31
getLicenseInfo()	33
getLiveStreams()	36
getLiveStreamStats()	37
getMsgCacheStats()	39
getNetStreams()	41
getNetStreamStats()	42
getRecordedStreams()	43
getRecordedStreamStats()	45
getScriptStats()	46
getServerStats()	47
getSharedObjects()	49
getSharedObjectStats()	50
getUsers()	52
getUserStats()	53
getVHosts()	54
getVHostStats()	55
ping()	57
reloadApp()	58

removeAdmin()	59
removeApp()	61
removeVHostAlias()	62
restartVHost()	63
setConfig()	64
setConfig2()	67
startServer()	70
startVHost()	71
stopServer()	72
stopVHost()	73
unloadApp()	74

# Administration API Reference

Use the Administration API to create tools that let you monitor and administer Adobe® Flash® Media Server 3.

**Note:** In previous versions of Flash Media Server, the Administration API was called the Server Management API.

You can call Administration API methods from client-side ActionScript 1.0, ActionScript 2.0, or ActionScript 3.0. To call methods over HTTP, add permissions to the Users.xml configuration file and build an HTTP request. To call methods over RTMP or RTMPE, call the client-side `NetConnection.call()` method and pass it the Administration API method, a response object, and any additional parameters.

For details about using the Administration API, see “Using the Administration API” in *Adobe Flash Media Server Configuration and Administration Guide*.

Entries in this document are listed alphabetically. Where the document refers to *RTMP/E*, it means RTMP or RTMPE.

## addAdmin()

**RTMP/E** `addAdmin(username:String, password:String [,scope:String]): Object`

### HTTP

`http://www.example.com:1111/admin/addAdmin?auser=aUsername&apswd=aPassword&user|username=username&pass|password=password  
[&scope=scope]`

Adds an administrator to the system. The administrator can be a server administrator or a virtual host administrator depending on which parameters you pass. If the administrator already exists, this method fails. Otherwise, the administrator is added and the Users.xml file is updated with the new values. (Virtual host administrators are added to the Users.xml file in a specific virtual host folder.)

You must be a server administrator to add server administrators or to add administrators for virtual hosts other than the one currently connected to.

### Availability

Flash Communication Server 1.0.

### Parameters

`user|username` A String indicating the user name of the administrator being added.

`pass|password` A String indicating the password of the administrator being added. The password is encoded before it is written to the Users.xml configuration file. If the `scope` parameter has a virtual host specified, the new administrator is added to the Users.xml file of the virtual host.

`aUsername` A String indicating the username of the administrator making the call.

**Note:** In the RTMP/E call, the username and password of the administrator making the call are passed in the `NetConnection.connect()` call.

`aPassword` A String indicating the password of the administrator making the call.

**scope** A String indicating whether the administrator is a server administrator or a virtual host administrator, and for which virtual host. To add a server administrator, specify `server`. To add an administrator to the virtual host to which you're connected, omit this parameter. To add a virtual host administrator to a different virtual host, specify the virtual host as `adaptor_name/virtual_host_name`.

**Note:** The parameter data types are relevant only for RTMP/E calls.

### Returns

**RTMP/E** If the call succeeds, the server sends an information object with a `level` property of `status` and a `code` property of `NetConnection.Call.Success`. If the call fails, the server sends an information object with a `level` property of `error` and a `code` property of `NetConnection.Admin.CommandFailed` or a more specific value, if available. The information object also includes a `timestamp` property indicating the date and time the call was made.

**HTTP** The call returns XML with the following structure:

```
<result>
  <level>
  <code>
  <timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

### Example

The following three examples show how you can specify parameters in a call to the `addAdmin()` method over RTMP:

```
nc_admin = new NetConnection();
nc_admin.connect("rtmp://localhost:1111/admin", "JLee", "x52z49ab");

/* Adds a server administrator named "GLee" with password "boat4907" */
nc_admin.call("addAdmin", new onAddAdmin(), "GLee", "boat4907", "server");

/* Adds a virtual host administrator named "ChrisM" with password "tree2981" */
nc_admin.call("addAdmin", new onAddAdmin(), "ChrisM", "tree2981");

/* Adds a virtual host administrator "DHong" with password "wate3235" */
/* for vhost tree.oak.com */
nc_admin.call("addAdmin", new onAddAdmin(), "DHong", "wate3235", "_defaultRoot_/" +
tree.oak.com");
```

## addApp()

**RTMP/E** `addApp(appname:String) : Object`

**HTTP** `http://www.example.com:1111/admin/addApp?auser=username&apswd=password`  
&app|appname=applicationname

Adds a new application to the virtual host you are connected to by creating the required directory for the new application in the directory tree. Once the directory for the new application is created, you (or another administrator with file system access) can put any required server-side scripts in the directory. The client-side code uses the new application directory in the URI parameter of the `NetConnection.connect` call.

A virtual host may have multiple application directories. This application is created in the first application directory specified in the XML configuration file.

### Availability

Flash Communication Server 1.0.

### Parameters

`app|appname` A String indicating the name of the application to be added.

`auser` A String indicating the user name of the administrator.

`apswd` A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server returns an Object with a `level` property of `status` and a `code` property of `NetConnection.Call.Success`.

If the call fails, the server returns an Object with a `level` property of `error` and a `code` property of `NetConnection.Admin.CommandFailed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** The call returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call. The XML may contain a `description` element if available.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

### Example

The following example shows a call to add the ChatApp application to the connected virtual host:

```
nc_admin = new NetConnection();
nc_admin.connect("rtmp://localhost:1111/admin", "JLee", "x52z49ab");
nc_admin.call("addApp", new onAddApp(), "ChatApp");
```

## addVHostAlias()

**RTMP/E** `addVHostAlias(vhost:String, alias:String, persist:Boolean) : Object`

**HTTP** `http://www.example.com:1111/admin/addVHostAlias?auser=username&apswd=password&vhost=hostname&alias=alias&persist=value`

Adds an alias to a virtual host. Aliases are alternative names for virtual hosts that are used as targets by incoming Flash Media Server connections. When you remove an alias, that name is no longer available for incoming connections.

### Availability

Flash Media Server 2.0.

**Parameters**

**vhost** A String indicating the virtual host to which to add an alias.

**alias** A String indicating the alias name to add to the specified virtual host.

**persist** A Boolean value indicating whether the alias change will be written to the configuration file to last beyond the virtual host's next restart (`true`), or whether this alias will be lost on virtual host restart (`false`).

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

**Returns**

**RTMP/E** If the call succeeds, the server returns an Object with a `level` property of `status` and a `code` property of `NetConnection.Call.Success`. If the call fails, the server returns an Object with a `level` property of `error` and a `code` property of `NetConnection.Admin.CommandFailed`.

**HTTP** A call returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

**Example**

```
nc_admin = new NetConnection();
nc_admin.connect("rtmp://localhost:1111/admin", "JLee", "x52z49ab");
nc_admin.call("addVHostAlias", new Result(), "myvhost", "myalias", true);
```

**See also**

[removeVHostAlias\(\)](#)

## approveDebugSession()

**RTMP/E** `approveDebugSession(appInst:String, PIN:Number) : Object`

**Note:** This method is not supported over HTTP.

Approves a pending debug session's request to connect to a selected application. Once approved and permitted, the debug session may connect to its application and act as a regular client. Connecting to an application with a debug session allows you to view the streams and shared objects for an application through the Administration Console.

**Availability**

Flash Media Server 2.0.

**Parameters**

**appInst** A String indicating the application and instance name that has a pending debug connection to approve.



**PIN** A number indicating the debug session personal identification number. Each debug connection issues a debug number when queueing to connect to an application. This same number is included on this API. When this API is processed, the PIN numbers are matched and the corresponding connection is allowed to connect. This is a security measure to prevent unauthorized users from using the debug connection.

Keep in mind that 0 is not a valid **PIN** value. To maintain security, PIN numbers should not be able to be easily guessed. PIN numbers must be non-zero and must be within the range of  $\pm (2^{31} - 1)$  that is between positive or negative 2147483648. If there is a pending connection with a given PIN, and another connection arrives with the same PIN while the original is still waiting, the second connection with the same PIN will be rejected as a security measure.

### Returns

**RTMP/E** If the call succeeds, the server returns an Object with a **level** property of **status** and a **code** property of **NetConnection.Call.Success**. If the call fails, the server returns an Object with a **level** property of **error** and a **code** property describing the failure.

### Example

The following is an example of a debug session:

```
nc_admin.connect("rtmp://serverName/appName%3F%5Ffcs%5Fdebugreq%5F%3D1234");
// The original string is _fcs_debugreq=1234.
```

The following is an example of a Debug approval request:

```
nc_admin.call("approveDebugSession", null, "appName/instName", 1234);
```

## changePswd()

**RTMP/E** `changePswd(admin_name:String, password:String [,scope:String]) : Object`

**HTTP** `http://www.example.com:1111/admin/changePswd?user=username&apswd=password&username=name &password=password[&scope=scope]`

Changes the password for the specified administrator. The password is encoded before it is written to the Users.xml configuration file.

Virtual host administrators can change only their own password.

### Availability

Flash Communication Server 1.0.

### Parameters

**admin\_name** A String indicating the name of the administrator whose password is being changed.

**username** A String indicating the name of the administrator whose password is being changed.

**password** A String indicating the administrator's new password.

**scope** A String indicating whether the administrator is a server administrator or virtual host administrator, and for which virtual host.

To change the password for the specified administrator on the virtual host to which you're connected, omit this parameter. To change the password for the specified administrator on a different virtual host, specify `adaptor_name/virtual_hostname`.

To change a server administrator's password, specify `server`.

`auser` A String indicating the user name of the administrator.

`apswd` A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status` and a `code` property of `NetConnection.Call.Success`.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Admin.CommandFailed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

If the specified administrator does not exist, this method fails.

**HTTP** A call returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

### Example

```
nc_admin = new NetConnection();
nc_admin.connect("rtmp://localhost:1111/admin", "JLee", "x52z49ab");

/* Change password for server administrator named "ASilva" to "cbx5978y" */
nc_admin.call("changePswd", new onChangePswd(), "ASilva", "cbx5978y", "server");

/* Change password for virtual host administrator "JLee" to "kbat3786" on */
/* virtual host "tree.oak.com" */
nc_admin.call("changePswd", new onChangePswd(), "JLee", "kbat3786", "_defaultRoot_/_tree.oak.com");
```

## gc()

**RTMP/E** `gc()` : Object

**HTTP** `http://www.example.com:1111/admin/gc?auser=username&apswd=password`

Forces collection and elimination of all server resources that are no longer used, such as closed streams, instances of applications, and nonpersistent shared objects. This operation is performed within about one second of the call.

You must be a server administrator to perform this operation.

### Availability

Flash Communication Server 1.0.

### Parameters

`auser` A String indicating the user name of the administrator.

`apswd` A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status` and a `code` property of `NetConnection.Call.Success`.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Admin.CommandFailed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** A call returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call. Some XML might also have a `description` element that contains a string describing the cause of the failure.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## getActiveInstances()

**RTMP/E** `getActiveInstances(processID:Number) : Object`

**HTTP** `http://www.example.com:1111/admin/getActiveInstances?auser=username&apswd=password&pid|processID=number`

Returns an array of strings that contains the names of all running application instances on the FMSCore process specified by the `processID` parameter.

### Availability

Flash Communication Server 1.0.

### Parameters

`pid|processID` A number indicating the process identifier of an FMSCore process.

`auser` A String indicating the user name of the administrator.

`apswd` A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an array of strings containing the names of all running instances of an application on the server or on a specified process.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Admin.CommandFailed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** A successful call returns XML with the following structure:

```
<result>
```

```

    <level></level>
    <code></code>
    <timestamp></timestamp>
    <data></data>
</result>

```

An unsuccessful call returns XML with the following structure:

```

<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>

```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

### Example

```

nc_admin = new NetConnection();
nc_admin.connect("rtmp://localhost:1111/admin", "JLee", "x52z49ab");
nc_admin.call("getActiveInstances", new Result());

```

## getActiveVHosts()

**RTMP/E** `getActiveVHosts([adaptor]);`

**HTTP** `http://www.example.com:1111/admin/getActiveVHosts?auser=username&apswd=password`  
`[&adaptor=name]`

Returns an array of active virtual hosts defined for the specified adaptor. You must be a server administrator to call this method. A virtual host is active if at least one application running on it is connected.

### Availability

Flash Media Server 3.

### Parameters

**adaptor** A String indicating the user name of the adaptor. If not specified, it is assumed to be `_defaultRoot_`.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an array of strings containing the names of all the active virtual hosts.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML like the following:

```

<result>

```

```
<level>status</level>
<code>NetConnection.Call.Success</code>
<timestamp>9/23/2007 6:16:40 PM</timestamp>
<data>
  <_0>
    <vhost_name>vhost1</vhost_name>
  </_0>
  <_1>
    <vhost_name>_defaultVHost_</vhost_name>
  </_1>
</data>
</result>
```

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## getActiveVHostStats()

**RTMP/E** getActiveVHostStats([verbose]:Boolean) : Object

**HTTP** http://www.example.com:1111/admin/getActiveVHostStats?auser=username&apswd=password [&verbose=value]

Returns aggregate performance data for all instances for all applications for the active virtual hosts on any active core processes. Whenever a virtual host functions among multiple cores, this method displays the statistics for the active core processes. You must be a server administrator to call this method.

### Availability

Flash Media Server 3.

### Parameters

**verbose** Boolean; `true` displays the statistics for the applications running on the active virtual host along with the aggregate statistics for the virtual host; `false` displays only the aggregate statistics of the active virtual host and not the statistics per virtual host. The default value is `true`.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object containing the active virtual host performance data. The `data` object has the following properties:

Property	Description
<code>msg_in</code>	Number; total number of messages processed by this virtual host.
<code>msg_out</code>	Number; total number of messages sent by this virtual host.
<code>msg_dropped</code>	Number; total number of messages dropped by this virtual host.
<code>bytes_in</code>	Number; total number of bytes read by this virtual host.
<code>bytes_out</code>	Number; total number of bytes written by this virtual host.

Property	Description
accepted	Number; total number connections accepted by this virtual host.
rejected	Number; total number of connections rejected by this virtual host.
connected	Number; total number of connections currently active.
total_apps	Number; total number of applications that have been created.
total_connects	Number; total number of connections to the server.
total_disconnects	Number; total number of disconnections from the server.
total_instances_loaded	Number; total number of instances that have been loaded.  This property does not represent the total number of active instances loaded. To get the number of active instances loaded, subtract the value of <code>total_instances_unloaded</code> from <code>total_instances_loaded</code> .
total_instances_unloaded	Number; total number of instances that have been unloaded.
bw_in	Number; current bandwidth in, in bytes per second.
bw_out	Number; current bandwidth out, in bytes per second.
tunnel_bytes_in	Number; total number of bytes read through the tunnel.
tunnel_bytes_out	Number; total number of bytes written through the tunnel.
tunnel_requests	Number; number of current requests.
tunnel_idle_requests	Number; number of currently idle requests.
tunnel_idle_responses	Number; number of currently idle responses.
normal_connects	Number; total number of normal connections.
virtual_connects	Number; total number of connections through a remote edge.
group_connects	Number; total number of remote edges that are connected.
service_connects	Number; total number of service connections.
service_requests	Number; total number of services requested.
admin_connects	Number; total number of administrator connections.
debug_connects	Number; total number of debug connections.
swf_verification_attempts	A counter of the number of SWF verification attempts made. Represents the total SWF verification credentials passed to the server for checking. There may be more than one credential presented per connection.
swf_verification_exceptions	A counter of the number of SWF verification exceptions made. Exceptions are allowed through explicit configuration in the Application.xml file that allows certain user agents to bypass the requirement for SWF verification. Every connection allowed as an exception is counted here.
swf_verification_failures	A counter of the number of SWF verification failures. Failures result from the presentation of SWF verification credentials that are found not to be a match for any loaded credential. Each failure corresponds to a disconnection of the presenting connection.

Property	Description
<code>swf_verification_unsupported_rejects</code>	A counter of the number of SWF verification unsupported rejections. When a version of Flash Player that doesn't support SWF verification connects to an application that requires SWF verification, the unsupported rejection count is increased and the connecting client is disconnected.
<code>swf_verification_matches</code>	A counter of the total number of matches. When an authentic SWF verification credential is presented, this number increases. There may be more than one match per connection.
<code>swf_verification_remote_misses</code>	A counter of the proxy/remote server process's missed SWF verification attempts. Whenever a proxy/remote server receives a SWF verification attempt it looks to its local cache for valid SWF verification. If it does not locate a match it logs a remote miss and defers to the origin to answer the verification attempt.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the data element are elements with each of the properties listed in the RTMP/E returns section.

If the call fails, it returns something like the following:

```
<result>
  <level>error</level>
  <code>Admin.API.InvalidMethod</code>
  <description>getactivevhoststatss - No such method found!</description>
  <timestamp>9/13/2007 7:55:58 PM</timestamp>
</result>
```

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## getAdaptors()

**RTMP/E** `getAdaptors()` : Object

**HTTP** `http://www.example.com:1111/admin/getAdaptors?auser=username&apswd=password`

Returns an array of adaptors that are defined. You must be a server administrator to call this method.

### Availability

Flash Communication Server 1.0.

### Parameters

`auser` A String indicating the user name of the administrator.

`apswd` A String indicating the password of the administrator.

## Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an array of strings containing the names of all the adaptors.

**HTTP** If the call succeeds, it returns something like the following:

```
<result>
  <level>status</level>
  <code>NetConnection.Call.Success</code>
  <timestamp>7/23/2007 8:25:52 AM</timestamp>
  <data>
    <_0>Adaptor2</_0>
    <_1>_defaultRoot_</_1>
  </data>
</result>
```

If the call fails, it returns something like the following:

```
<result>
  <level>error</level>
  <code>NetConnection.Connect.Rejected</code>
  <description>Administrator login failed for user Admin.</description>
  <timestamp>7/23/2007 8:26:36 AM</timestamp>
</result>
```

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## Example

```
nc_admin = new NetConnection();
nc_admin.connect("rtmp://localhost:1111/admin", "JLee", "x52z49ab");
nc_admin.call("getAdaptors", new Result());
```

# getAdminContext()

**RTMP/E** `getAdminContext([adminName:String][, adaptorName:String][, vhostName:String])` : Object

**HTTP** `http://www.example.com:1111/admin/getAdminContext?user=username&pswd=password [&adminName=name] [&adaptorName=name] [&vhostName=name]`

Gets the administrative context for an administrator, including information about the specified user's administrative permissions, the name of the adaptor and virtual host to which the user is connected, and whether the user is currently connected to Flash Media Server.

## Availability

Flash Communication Server 1.0.

## Parameters

**adminName** A String indicating the name of an administrator.

**adaptorName** A String indicating an alternate adaptor, other than `_defaultRoot_`, on which to find administrators. If not specified, `_defaultRoot_` is used.



**vhostName** A String indicating an alternate virtual host, other than `_defaultVHost_`, on which to find virtual host administrators. If not specified, server level admin is used.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object. The data object has the following properties:

Property	Description
<code>admin_type</code>	String; the type of administrator, either <code>server</code> or <code>virtual host</code> .
<code>adaptor</code>	String; name of the adaptor for which the user is an administrator.
<code>vhost</code>	String; name of the virtual host for which the user is an administrator.
<code>connected</code>	String; this property is deprecated and always returns <code>true</code> .

**HTTP** If the call succeeds, it returns something like the following:

```
<result>
  <level>status</level>
  <code>NetConnection.Call.Success</code>
  <timestamp>7/23/2007 8:57:26 AM</timestamp>
  <data>
    <admin_type>server</admin_type>
    <adaptor>_defaultRoot_</adaptor>
    <vhost>_defaultVHost_</vhost>
    <connected>true</connected>
  </data>
</result>
```

If the call fails, it returns something like the following:

```
<result>
  <level>error</level>
  <code>NetConnection.Connect.Rejected</code>
  <description>Administrator login failed for user Admin.</description>
  <timestamp>7/23/2007 9:00:44 AM</timestamp>
</result>
```

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## getAdmins()

**RTMP/E** `getAdmins(adaptorName:String) : Object`

**HTTP** `http://www.example.com:1111/admin/getAdmins?auser=username&apswd=password&adaptorName=name`

Returns an array of all administrators on Flash Media Server. The returned data has two top-level groups: server-level administrators and virtual host-level administrators. The server-level administrators group is a simple list of names. The virtual host administrators are subdivided by virtual host and each virtual host contains a list of administrator names.

### Availability

Flash Media Server 2.0.

### Parameters

**adaptorName** A String indicating an alternate adaptor, other than `_defaultRoot_`, on which to find virtual host administrators. If not specified, `_defaultRoot_` is used.

**ouser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an array. The `data` object has the following properties:

Property	Description
<code>server_admins</code>	Array; an array of administrator names.
<code>admin</code>	String; name of an administrator.
<code>vhost_admins</code>	Array; an array of virtual host names.
<code>admin</code>	String; name of the virtual host for which the user is an administrator.

**HTTP** If the call succeeds, it returns something like the following:

```
<result>
  <level>status</level>
  <code>NetConnection.Call.Success</code>
  <timestamp>7/23/2007 9:05:00 AM</timestamp>
  <data>
    <server_admins>
      <admin>Admin</admin>
      <admin>John</admin>
      <admin>Mary</admin>
    </server_admins>
    <vhost_admins>
      <_defaultVHost_>
        <_0>Kent</_0>
      </_defaultVHost_>
    </vhost_admins>
  </data>
</result>
```

If the call fails, it returns something like the following:

```
<result>
  <level>error</level>
  <code>NetConnection.Connect.Rejected</code>
  <description>Administrator login failed for user Admin.</description>
  <timestamp>7/23/2007 9:08:25 AM</timestamp>
</result>
```

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## getApps()

**RTMP/E** `getApps([verbose:Boolean] [, force:Boolean]):Object`

**HTTP** `http://www.example.com:1111/admin/getApps?auser=username&apswd=password [&verbose=true|false [&force=true|false]]`

Returns an array of strings that contains the names of all the applications that are installed. Calling `addApp()` or `removeApp()` refreshes the cached application list.

**Note:** Applications that are added or removed using the file system are not reflected in the cached list.

### Availability

Flash Communication Server 1.0. The verbose and force parameters are available in Flash Media Server 3.

### Parameters

**verbose** Boolean; `true` displays all the applications under a virtual host; `false` displays the total number of applications. The default value is `true`.

**force** Boolean; `true` forces a refresh of the cached list of applications and retrieves the list; `false` retrieves a cached list of applications. If you want to force a refresh, you must specify both the `verbose` and `force` parameters. The default value is `false`.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an array of strings containing the names of all the applications that are installed.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level>status</level>
  <code>NetConnection.Call.Success</code>
  <timestamp>7/23/2007 9:10:37 AM</timestamp>
  <data>
    <total_apps>3</total_apps>
    <_0>live</_0>
    <_1>livetest</_1>
    <_2>vod</_2>
  </data>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call. Some XML might also have a `description` element that contains a string describing the cause of the failure.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

#### See also

[getActiveInstances\(\)](#), [getAppStats\(\)](#), [getInstanceStats\(\)](#)

## getAppStats()

**RTMP/E** `getAppStats(appname:String):Object`

**HTTP** `http://www.example.com:1111/admin/getAppStats?auser=username&apswd=password&app|appname=name`

Gets aggregate performance data for all instances of the specified application.

#### Availability

Flash Communication Server 1.0.

#### Parameters

**appname** A String indicating the name of the application to be added.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

#### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object. The `data` object has the following properties:

Property	Description
<code>accepted</code>	Number; total number of connection attempts accepted by this application.
<code>bytes_in</code>	Number; total number of bytes read by this application.
<code>bytes_out</code>	Number; total number of bytes written by this application.
<code>connected</code>	Number; total number of connections currently active.
<code>launch_time</code>	ActionScript Date object; time the application started.
<code>msg_dropped</code>	Number; total number of messages dropped by this application.
<code>msg_in</code>	Number; total number of messages processed by this application.
<code>msg_out</code>	Number; total number of messages sent by this application.
<code>normal_connects</code>	Number; total number of normal connections.
<code>virtual_connects</code>	Number; total number of connections through a remote edge.
<code>group_connects</code>	Number; total number of remote edges that are connected.
<code>service_connects</code>	Number; total number of service connections.
<code>service_requests</code>	Number; total number of services requested.

Property	Description
admin_connects	Number; total number of administrator connections.
debug_connects	Number; total number of debug connections.
rejected	Number; total number of connection attempts rejected by this application.
bw_in	Number; current bandwidth in, in bytes per second.
bw_out	Number; current bandwidth out, in bytes per second.
total_connects	Number; total number of socket connections to the application since the application was started.
total_disconnects	Number; total number of disconnections from the application since the application was started.
total_instances_loaded	Number; total number of instances that have been loaded since the application started.  This property does not represent the total number of active instances loaded. To get the number of active instances loaded, subtract the value of <code>total_instances_unloaded</code> from <code>total_instances_loaded</code> .
total_instances_unloaded	Number; total number of instances that have been unloaded since the application started.
up_time	Number; time, in seconds, the application has been running.
swf_verification_attempts	A counter of the number of SWF verification attempts made. Represents the total SWF verification credentials passed to the server for checking. There may be more than one credential presented per connection.
swf_verification_exceptions	A counter of the number of SWF verification exceptions made. Exceptions are allowed through explicit configuration in the Application.xml file that allows certain user agents to bypass the requirement for SWF verification. Every connection allowed as an exception is counted here.
swf_verification_failures	A counter of the number of SWF verification failures. Failures result from the presentation of SWF verification credentials that are found not to be a match for any loaded credential. Each failure corresponds to a disconnection of the presenting connection.
swf_verification_unsupported_rejects	A counter of the number of SWF verification unsupported rejections. When a version of Flash Player that doesn't support SWF verification connects to an application that requires SWF verification, the unsupported rejection count is increased and the connecting client is disconnected.
swf_verification_matches	A counter of the total number of matches. When an authentic SWF verification credential is presented, this number increases. There may be more than one match per connection.
swf_verification_remote_misses	A counter of the proxy/remote server process's missed SWF verification attempts. Whenever a proxy/remote server receives a SWF verification attempt it looks to its local cache for valid SWF verification. If it does not locate a match it logs a remote miss and defers to the origin to answer the verification attempt.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, the server returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

### Example

The following two examples use an application instance named `app_1` on both an origin and an edge server. Use the following URL to call `getAppStats()` on an origin server:

```
http://origin.example.com:1111/admin/getAppStats?auser=username&apswd=password
&appName=app_1
```

Use the following URL to call `getAppStats()` on an edge server:

```
http://edge.example.com:1111/admin/getAppStats?auser=username&apswd=password&appName=app_1
@edge.example.com
```

**Note:** To obtain the application name, log in to the edge server on the Administration Console or call `getActiveInstances()`. When `appName=app_1` is run on the edge server, all zeroed information is displayed.

### See also

[getApps\(\)](#), [getInstanceStats\(\)](#)

## getConfig()

**RTMP/E** `getConfig(key:String [,scope:String])` : Object

**HTTP** `http://www.example.com:1111/admin/getConfig?auser=username&apswd=password
&key=configkey[&scope=scope]`

This API has been deprecated; use [getConfig2\(\)](#) instead.

Gets the value of a configuration parameter in a configuration file.

Virtual host administrators can view configuration parameters in the `Vhost.xml` file and `Application.xml` files for their own virtual hosts. You must be a server administrator to view most of the configuration parameters for the `Server.xml` and `Adaptor.xml` files. For a description of the XML configuration files, see *Adobe Flash Media Server Configuration and Administration Guide*.

### Availability

Flash Communication Server 1.0.

### Parameters

**key** A String indicating the configuration parameter for which information is retrieved.

A key is specified as a list of subkeys that are delimited by slashes (/). The first subkey specifies the XML configuration file that contains the desired configuration parameter. Subsequent subkeys correspond to tags that are relative to the XML configuration file; the hierarchy and names of the subkeys match the tags in the XML file.

Depending on your permissions, you can get configuration parameters for the following files:

- For the `Server.xml` file, specify `Admin` or `Server` as the first subkey. All subsequent keys correspond to tags that are relative to the `Admin` or `Server` tag in the `Server.xml` file.

You must be a server administrator to view configuration parameters in the `Server` tag.

Virtual host administrators can view configuration parameters in the `Admin` tag for their own virtual host only. They might not be able to view certain kinds of sensitive information; for example, they can view the names of other administrators for their own virtual host, but they cannot view those administrators' passwords or permission settings.

- For the `Adaptor.xml` file, specify as the first subkey `Adaptor:adaptor_name`, where `adaptor_name` is the name of the adaptor. All subsequent parameters correspond to keys that are relative to the `Adaptor` tag in the `Adaptor.xml` file.
- For the `Vhost.xml` file, specify as the first subkey `Adaptor:adaptor_name/VirtualHost:vhost_name`, where `adaptor_name` is the name of the adaptor and `vhost_name` is the name of the virtual host. All subsequent keys correspond to tags that are relative to the `VirtualHost` tag in the `Vhost.xml` file.
- For the `Application.xml` file of an application that is running on the same virtual host to which you connected when you logged on to the administration server, specify as the first subkey `Application:app_name`, where `app_name` is the name of the application.

To get a parameter in the `Application.xml` file for an application that is running on a different virtual host, specify the full key `Adaptor:adaptor_name/VirtualHost:vhost_name/Application:app_name`. You must also specify the `scope` parameter.

To get the default `Application.xml` file, specify `Application` without the colon (:) and the `app_name` attribute.

**scope** A String. To get a configuration parameter in the `Server.xml` file, `Adaptor.xml` file, or `Vhost.xml` file, specify a slash (/).

To get a configuration parameter in the `Application.xml` file for an application that is running on the same virtual host to which you connected when you logged on to Flash Media Server, omit this parameter.

**Note:** To determine the adaptor or virtual host to which you're connected, use the `getAdminContext()` method.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

## Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` object whose properties contain the values of the specified tag.

If the call fails (that is, if the specified configuration parameter isn't found), the server returns an empty string.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## Examples

The following examples show how to get configuration keys in each of the four XML files:

```
// Establish connection to server.
nc_admin = new NetConnection();
nc_admin.connect("rtmp://localhost:1111/admin", "JLee", "xm1472dy");

// For a virtual host administrator, find key in Server.xml.
key = "Admin/Server/UserList/User:JLee/Password";
nc_admin.call("getConfig", new onGetConfig(), key, "/");

// For a server administrator; find key in Server.xml.
key = "Server/LicenseInfo";
nc_admin.call("getConfig", new onGetConfig(), key, "/");

// Find key in Adaptor.xml.
key = "Adaptor:_defaultRoot_/HostPortList/HostPort";
nc_admin.call("getConfig", new onGetConfig(), key, "/");

// Find key in Vhost.xml.
key = "Adaptor:_defaultRoot_/VirtualHost:_defaultVhost_/RecordAccessLog";
nc_admin.call("getConfig", new onGetConfig(), key, "/");

// Find key in Application.xml for an application on the virtual host you
// connected to when you logged on to the administration server.
// Note that the previous subkeys and the second parameter "/" are not necessary.
key = "Application:FinanceApp/RecordAppLog";
nc_admin.call("getConfig", new onGetConfig(), key);

// Find key in Application.xml for an application on a different virtual host.
key = "Adaptor:_defaultRoot_/VirtualHost:www.redpin.com/Application:ChatApp/ -
RecordAppLog";
nc_admin.call("getConfig", new onGetConfig(), key, "/");
```

## See also

[getAdminContext\(\)](#), [setConfig\(\)](#)

# getConfig2()

**RTMP/E** getConfig2(key:String, scope:String) : Object

**HTTP** http://www.example.com:1111/admin/getConfig2?auser=username&apswd=password  
&key=configkeyname&scope=scope



Gets the value of a configuration parameter in a specified configuration file. Flash Media Server has six server configuration files from which you can retrieve information: Users.xml, Logger.xml, Server.xml, Adaptor.xml, Vhost.xml, and Application.xml. For a description of the XML configuration files, see *Adobe Flash Media Server Configuration and Administration Guide*.

Virtual host administrators can view configuration parameters in the Vhost.xml file and Application.xml files for their own virtual hosts. You must be a server administrator to view most of the configuration parameters for the Server.xml and Adaptor.xml files.

**Note:** It is possible to have more than one XML tag with the same name at the same level in the XML tree. In the configuration file, you should distinguish such tags by using a name attribute in the XML tag (for example, if you have more than one VirtualHost tag: `<VirtualHost name="www.redpin.com"></VirtualHost>`). When you call the `getConfig()` method and specify the configuration subkeys, you can indicate which tag you want by specifying the tag name, followed by a colon and the correct name attribute, for example, `Admin/Adaptor:_defaultRoot_/VirtualHost:www.redpin.com`.

### Availability

Flash Media Server 2.0.

### Parameters

**key** A String indicating the configuration parameter for which information is retrieved.

A key is specified as a list of subkeys that are delimited by slashes (/). The first subkey specifies the XML configuration file that contains the desired configuration key. Subsequent subkeys correspond to tags that are relative to the XML configuration file; the hierarchy and names of the subkeys match the tags in the XML file. If multiple tags exist with the same name and same parent, they can be distinguished by specifying a name attribute and appending the name attribute to the tag name separated by a colon in the key parameter. If the specified tag is a leaf node, then its tag data is returned. If the specified tag is not a leaf node, the whole tag is returned as an XML string.

**scope** A String indicating which configuration file to search for the configuration tag specified in the key parameter.

Flash Media Server has six server configuration files: Server.xml, Users.xml, Logger.xml, Adaptor.xml, Vhost.xml, and Application.xml. Depending on your permissions, you can get configuration keys for all these files, as described in the following list:

- / specifies Server.xml.
- Users specifies Users.xml for server administrators.
- Logger specifies Logger.xml.
- Adaptor:<adaptor\_name> specifies Adaptor.xml. Specify the adaptor name in place of the <adaptor\_name> placeholder. You must have server administrator privileges to access the Adaptor.xml file. If <adaptor\_name> is not the name of the adaptor the caller is connected to, the call fails.
- Adaptor:<adaptor\_name>/VHost:<vhost\_name> specifies VHost.xml. Specify the virtual host name name in place of the <vhost\_name> placeholder. If <adaptor\_name> is not the name of the adaptor the caller is connected to, or <vhost\_name> is not the name of the virtual host that the caller is connected to, the call fails.
- Adaptor:<adaptor\_name>/VHost:<vhost\_name>/Users.xml specifies Users.xml for virtual host administrators.

- `Adaptor:<adaptor_name>/VHost:<vhost_name>/App[:<app_name>]` specifies `Application.xml`. If no `<app_name>` is specified, the default `Application.xml` file is assumed. Otherwise, the application-specific `Application.xml` for the specified application is used. If the specified application is not defined, or the application does not have an application-specific `Application.xml` file, the call fails.

**Note:** To determine the adaptor or virtual host to which you're connected, call the `getAdminContext()` method.

`auser` A String indicating the user name of the administrator.

`apswd` A String indicating the password of the administrator.

## Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` object whose properties contain information about the specified tag.

If the specified tag is a leaf node, the call returns the tag data. Otherwise, the call returns the tag itself. For example, if the desired tag is `<foo>bar</foo>`, the call returns `"bar"`. However, if the desired tag contains child tags such as `<foo><bar>foobar</bar></foo>`, the call returns `"<foo><bar>foobar</bar></foo>"`.

If the call fails (that is, if the specified configuration key isn't found), the server returns an empty string.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## Examples

The following examples get XML data from various configuration files:

```
tSocket = new NetConnection();
tSocket.connect("rtmp://localhost/admin", "user", "password");

// find key in Server.xml
key = "Server/LicenseInfo";
tSocket.call("getConfig2", new onGetConfig(), key, "/");

// find key in Adaptor.xml
key = "HostPortList/HostPort";
scope = "Adaptor:_defaultRoot_";
```

```

tSocket.call("getConfig2", new onGetConfig(), key, scope);

// find key in Vhost.xml
key = "AppsDir";
scope = "Adaptor:_defaultRoot_/VHost:_defaultVHost_";
tSocket.call("getConfig2", new onGetConfig(), key, scope);

// find key in Application.xml for app "foo"
key = "Process/Scope";
scope = "Adaptor:_defaultRoot_/VHost:_defaultVHost_/App:foo";
tSocket.call("getConfig2", new onGetConfig(), key, scope);

// find key in default Application.xml
key = "Process/Scope";
scope = "Adaptor:_defaultRoot_/VHost:_defaultVHost_/App";
tSocket.call("getConfig2", new onGetConfig(), key, scope);

// return the whole Vhost.xml
key = "";
scope = "Adaptor:_defaultRoot_/VHost:_defaultVHost_";
tSocket.call("getConfig2", new onGetConfig(), key, scope);

```

**See also**

[getAdminContext\(\)](#), [setConfig2\(\)](#)

## getFileCacheStats()

**RTMP/E** `getFileCacheStats([verbose:Boolean] [, file:String]) : Object`

**HTTP** `http://www.example.com:1111/admin/getFileCacheStats?auser=username&apswd=password  
[&verbose=value] [&file=string]`

Obtains data about the file cache including size of cache, number of file objects held in cache, and number of segment hits and misses. If the `verbose` parameter is set to `true`, individual statistics for each object in the cache are returned.

**Availability**

Flash Media Server 3.

**Parameters**

**verbose** Boolean; `true` displays individual statistics for each object in the cache; `false` displays a summary of all the objects in the cache. The default value is `false`.

**file** A String indicating the statistics of a file. You must specify a value in the form `[object_name] value`.

The `file` property value requires the following the syntax, "`flv: [path]`", "`mp3: [path]`", "`mp4: [path]`", and so on. To determine this value, first call the API with `verbose=true` and grab the path from the result, as in the following:

```

<object_name>5888_C:\Program Files\Adobe\Flash Media Server
3\applications\app_1\streams\_definst_\Coral_Reef_Adventure_spark_150k_105s</object_name>

```

**Note:** Remove `5888_` when it is used in the `file` property; only the file path should be used.

Do not use the `verbose` and `file` parameters at the same time. If both are used, the first one is taken and the second one is ignored. If there are many files in the cache, using `verbose` and setting it to `true` can be time-consuming. It may also cause a browser timeout if you call the API from a web browser. To reduce the amount of data being returned, pass the name of a single object as a parameter; this returns only the statistics for that object.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object. The `data` object has the following properties:

Property	Description
<code>num_objs</code>	Number; total number of file objects held in the cache.
<code>hits</code>	Number; total number of segment "hits" since the server started.
<code>misses</code>	Number; total number of segment misses since the server started.
<code>released</code>	Number; total number of segments released since the server started.
<code>bytes</code>	Number; current size of the cache, in bytes.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

If the `verbose` parameter is set to `true`, the following individual statistics are given for each object in the cache:

Property	Description
<code>object_name</code>	Name of the stream object.
<code>num_segments</code>	Number; total number of segments from this object currently cached.
<code>hits</code>	Hits for this object.
<code>misses</code>	Misses for this object.
<code>useCount</code>	Number; total number of clients accessing this object (including internal server clients).
<code>released</code>	Number; total number of segments released from this object.
<code>recording</code>	If this file is being recorded.
<code>bytes</code>	Number; size of this object in the cache.
<code>num_kfs</code>	Number; total number of frames in the keyframe cache for this object.
<code>kfbytes</code>	Number; size of the keyframe cache for this object, in bytes.

**HTTP** If the call succeeds, it returns XML with the following structure (the data is sample data):

```
<result>
  <level>status</level>
  <code>NetConnection.Call.Success</code>
  <timestamp>9/10/2007 4:25:39 PM</timestamp>
  <data>
    <num_objs>4</num_objs>
    <hits>5</hits>
    <misses>4</misses>
    <bytes>1048532</bytes>
    <released>0</released>
    <objects>
```

```

    <_0>
      <object_name>2712_C:\Program Files\Adobe\Flash Media Server
        3\applications\video\streams\_definst\_skatedog</object_name>
      <num_segments>4</num_segments>
      <hits>5</hits>
      <misses>4</misses>
      <useCount>2</useCount>
      <released>0</released>
      <recording>>false</recording>
      <bytes>1048532</bytes>
      <num_kfs>0</num_kfs>
      <kfbytes>0</kfbytes>
    </_0>
  </objects>
</data>
</result>

```

If the call fails, it returns XML with the following structure:

```

<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>

```

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

#### See also

[getAppStats\(\)](#), [getGroupStats\(\)](#)

## getGroupMembers()

**RTMP/E** `getGroupMembers(appInst:String, groupid:String) : Object`

**HTTP** `http://www.example.com:1111/admin/getGroupMembers?auser=username&apswd=password  
&appInst=name&groupid=string`

Returns a list of the group members for a particular group. Groups are multiplexed connections from a remote edge server to an origin server. Each group connection represents at least one individual connection to another Flash Media Server that is acting as an edge server for this server.

Call [getGroups\(\)](#) to get a value for the `groupid` parameter.

#### Availability

Flash Media Server 2.0.

#### Parameters

**appInst** A String indicating the name of the instance of the application on which the group resides, in the form `application_name/instance_name`. You must specify both the application name and the instance name, separated by a slash (/), even if you want performance statistics for the default instance of the application. For example, to specify the default instance for an application named `ChatApp`, specify `ChatApp/_defInst_`.

**groupid** A String indicating the group's client ID.

**auser** A String indicating the user name of the administrator.

`apswd` A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an array containing the client IDs of all the individual clients connected through this group.

**HTTP** If the call succeeds, it returns XML like the following:

```
<result>
  <level>status</level>
  <code>NetConnection.Call.Success</code>
  <timestamp>10/23/2007 12:19:24 PM</timestamp>
  <data>
    <name>_defaultRoot_: _defaultVHost_:: _2</name>
    <_0>DDAAQMSI</_0>
  </data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level>error</level>
  <code>NetConnection.Call.BadValue</code>
  <timestamp>10/23/2007 1:59:00 PM</timestamp>
  <description>Invalid group ID (0).</description>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The `timestamp` response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

### See also

[getGroups\(\)](#), [getGroupStats\(\)](#)

## getGroupStats()

**RTMP/E** `getGroupStats(appInst:String, groupid:String) : Object`

**HTTP** `http://www.example.com:1111/admin/getGroupStats?ouser=username&apswd=password  
&appInst=name&groupid=string`

Gets statistics for a particular group connection. This connection is special because it multiplexes for more than one connection and contains a unique statistic called `members_count`. Group connections are established from one server to another as proxies.

You can call [getGroups\(\)](#) to get a value for `groupid`.

### Availability

Flash Media Server 2.0.

## Parameters

**appInst** A String indicating the name of the instance of the application on which the group resides, in the form `application_name/instance_name`. You must specify both the application name and the instance name, separated by a slash (/), even if you want performance statistics for the default instance of the application. For example, to specify the default instance for an application named `ChatApp`, specify `ChatApp/_defaultInst_`.

**groupid** A String indicating the group's client ID.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

## Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object containing the client performance data. The `data` object has the following properties:

Property	Description
<code>connect_time</code>	ActionScript Date object; time the application connected to the server.
<code>protocol</code>	String; protocol used by the client to connect to the server ( <code>rtmp</code> , <code>rtmpe</code> , or <code>rtmpt</code> ).
<code>msg_in</code>	Number; total number of messages processed by this application.
<code>msg_out</code>	Number; total number of messages sent by this application.
<code>msg_dropped</code>	Number; total number of messages dropped by this application.
<code>bytes_in</code>	Number; total number of bytes read by this application.
<code>bytes_out</code>	Number; total number of bytes written by this application.
<code>msg_queue</code>	Object; client message queue statistics.
<code>total_queues</code>	Number; total number of queues for this client.
<code>audio</code>	Number; total number of audio messages in all audio queues.
<code>video</code>	Number; total number of video messages in all video queues.
<code>other</code>	Number; total number of cmd/data messages in the "other" queue.
<code>stream_ids</code>	Array; an array of numbers (stream IDs).
<code>members_count</code>	Number; the number of clients multiplexing on this group connection.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level>status</level>
  <code>NetConnection.Call.Success</code>
  <timestamp>10/23/2007 12:43:47 PM</timestamp>
  <data>
    <name>_defaultRoot_:defaultVHost::_:2</name>
    <bytes_in>3283</bytes_in>
    <bytes_out>3878</bytes_out>
    <msg_in>5</msg_in>
    <msg_out>1</msg_out>
    <msg_dropped>0</msg_dropped>
    <connect_time>10/23/2007 12:16:08 PM</connect_time>
    <protocol>rtmp</protocol>
```

```

    <msg_queue>
      <total_queues>1</total_queues>
      <audio>0</audio>
      <video>0</video>
      <other>0</other>
    </msg_queue>
    <stream_ids />
    <members_count>1</members_count>
  </data>
</result>

```

Nested in the data element are elements for each property of the data object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```

<result>
  <level>error</level>
  <code>NetConnection.Call.BadValue</code>
  <timestamp>10/23/2007 3:30:12 PM</timestamp>
  <description>Invalid group ID (0).</description>
</result>

```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

#### See also

[getGroups\(\)](#), [getGroupMembers\(\)](#)

## getGroups()

**RTMP/E** `getGroups(appInst:String) : Object`

**HTTP** `http://www.example.com:1111/admin/getGroups?auser=username&apswd=password&appInst=name`

Returns a list of the group connections for a particular application instance. Groups are multiplexed connections from a remote edge server to an origin server. Each group connection represents at least one individual connection to another Flash Media Server that is acting as an edge for this server.

#### Availability

Flash Media Server 2.0.

#### Parameters

**appInst** A String indicating the name of the instance of the application for which you want performance statistics, in the form `application_name/instance_name`. You must specify both the application name and the instance name, separated by a slash (/), even if you want performance statistics for the default instance of the application. For example, to specify the default instance for an application named `ChatApp`, specify `ChatApp/_defInst_`.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.



## Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an array of the client IDs of all groups connected to this application.

**HTTP** If the call succeeds, it returns XML like the following:

```
<result>
  <level>status</level>
  <code>NetConnection.Call.Success</code>
  <timestamp>7/24/2007 9:08:59 AM</timestamp>
  <data>
    <name>_defaultRoot_:_defaultVHost_::_2</name>
    <_0>CCAoQVeF</_0>
  </data>
</result>
```

If the call fails, it returns XML like the following:

```
<result>
  <level>error</level>
  <code>NetConnection.Admin.CommandFailed</code>
  <timestamp>7/24/2007 9:10:11 AM</timestamp>
  <description>Application instance is not loaded : vo</description>
</result>
```

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## See also

[getGroupMembers\(\)](#), [getGroupStats\(\)](#)

# getInstanceStats()

**RTMP/E** `getInstanceStats(appInst:String) : Object`

**HTTP** `http://www.example.com:1111/admin/getInstanceStats?auser=username&apswd=password&appInst=name`

Gets the performance data for a specified instance of an application.

If you only need information about the performance of a specific script, use the `getScriptStats()` method.

## Availability

Flash Communication Server 1.0.

## Parameters

**appInst** A String indicating the name of the instance of the application for which you want performance statistics, in the form `application_name/instance_name`. You must specify both the application name and the instance name, separated by a slash (/), even if you want performance statistics for the default instance of the application. For example, to specify the default instance for an application named `ChatApp`, specify `ChatApp/_defInst_`.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

## Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object. The `data` object has the following properties:

Property	Description
<code>launch_time</code>	ActionScript Date object; time the instance was launched.
<code>up_time</code>	Number; length of time, in seconds, the instance has been running.
<code>msg_in</code>	Number; total number of messages processed by this instance of the application.
<code>msg_out</code>	Number; total number of messages sent by this instance of the application.
<code>msg_dropped</code>	Number; total number of messages dropped by this instance of the application.
<code>bytes_in</code>	Number; total number of bytes read by this instance of the application.
<code>bytes_out</code>	Number; total number of bytes written by this instance of the application.
<code>accepted</code>	Number; total number of connection attempts accepted by this application.
<code>rejected</code>	Number; total number of connection attempts rejected by this application.
<code>connected</code>	Number; total number of connections currently active.
<code>total_connects</code>	Number; total number of socket connections to this instance of the application since the instance was started.
<code>total_disconnects</code>	Number; total number of socket disconnections from this instance of the application since the instance was started.
<code>script</code>	Object that contains script engine performance data. The following are properties of the script object:  <code>time_high_water_mark</code> : Number; maximum amount of time, in seconds, that the script has taken to execute an event.  <code>queue_size</code> : Number; total number of events currently in the script engine queue.  <code>total_processed</code> : Number; total number of events processed by the script engine.  <code>total_process_time</code> : Number; number of seconds taken to process the number of events in <code>total_processed</code> .  <code>queue_high_water_mark</code> : Number; maximum number of events in the queue.
<code>normal_connects</code>	Number; total number of normal connections.
<code>virtual_connects</code>	Number; total number of connections through a remote edge.
<code>group_connects</code>	Number; total number of remote edges that are connected.
<code>service_connects</code>	Number; total number of service connections.
<code>service_requests</code>	Number; total number of services requested.
<code>admin_connects</code>	Number; total number of administrator connections.
<code>debug_connects</code>	Number; total number of debug connections.
<code>swf_verification_attempts</code>	A counter of the number of SWF verification attempts made. Represents the total SWF verification credentials passed to the server for checking. There may be more than one credential presented per connection.

Property	Description
<code>swf_verification_exceptions</code>	A counter of the number of SWF verification exceptions made. Exceptions are allowed through explicit configuration in the Application.xml file that allows certain user agents to bypass the requirement for SWF verification. Every connection allowed as an exception is counted here.
<code>swf_verification_failures</code>	A counter of the number of SWF verification failures. Failures result from the presentation of SWF verification credentials that are found not to be a match for any loaded credential. Each failure corresponds to a disconnection of the presenting connection.
<code>swf_verification_unsupported_rejects</code>	A counter of the number of SWF verification unsupported rejections. When a version of Flash Player that doesn't support SWF verification connects to an application that requires SWF verification, the unsupported rejection count is increased and the connecting client is disconnected.
<code>swf_verification_matches</code>	A counter of the total number of matches. When an authentic SWF verification credential is presented, this number increases. There may be more than one match per connection.
<code>swf_verification_remote_misses</code>	A counter of the proxy/remote server process's missed SWF verification attempts. Whenever a proxy/remote server receives a SWF verification attempt it looks to its local cache for valid SWF verification. If it does not locate a match it logs a remote miss and defers to the origin to answer the verification attempt.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the data object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The `timestamp` response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

See also

[getActiveInstances\(\)](#), [getAppStats\(\)](#), [getScriptStats\(\)](#)

## getIOStats()

**RTMP/E** `getIOStats()` : Object

**HTTP** <http://www.example.com:1111/admin/getIOStats?user=username&passwd=password>

Returns detailed information about the network I/O characteristics of the connected adaptor.

You must be a server administrator to perform this operation.

### Availability

- Flash Communication Server 1.0.

### Parameters

`ouser` A String indicating the user name of the administrator.

`apswd` A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object. The data object has the following properties:

Property	Description
<code>msg_in</code>	Number; total number of messages processed by the adaptor.
<code>msg_out</code>	Number; total number of messages sent by the adaptor.
<code>bytes_in</code>	Number; total number of bytes read by the adaptor.
<code>bytes_out</code>	Number; total number of bytes written by the adaptor.
<code>reads</code>	Number; total number of system read calls.
<code>writes</code>	Number; total number of system writes.
<code>connected</code>	Number; number of currently active socket connections to the adaptor.
<code>total_connects</code>	Number; total number of socket connections to the adaptor since the adaptor was started.
<code>total_disconnects</code>	Number; total number of socket disconnections from the adaptor.
<code>msg_dropped</code>	Number; total number of messages dropped.
<code>tunnel_responses</code>	Number; number of tunneling responses thus far.
<code>normal_connects</code>	Number; total number of normal connections.
<code>virtual_connects</code>	Number; total number of connections through a remote edge.
<code>group_connects</code>	Number; total number of remote edges that are connected.
<code>service_connects</code>	Number; total number of service connections.
<code>service_requests</code>	Number; total number of services requested.
<code>admin_connects</code>	Number; total number of administrator connections.
<code>debug_connects</code>	Number; total number of debug connections.
<code>swf_verification_attempts</code>	A counter of the number of SWF verification attempts made. Represents the total SWF verification credentials passed to the server for checking. There may be more than one credential presented per connection.
<code>swf_verification_exceptions</code>	A counter of the number of SWF verification exceptions made. Exceptions are allowed through explicit configuration in the Application.xml file that allows certain user agents to bypass the requirement for SWF verification. Every connection allowed as an exception is counted here.

Property	Description
<code>swf_verification_failures</code>	A counter of the number of SWF verification failures. Failures result from the presentation of SWF verification credentials that are found not to be a match for any loaded credential. Each failure corresponds to a disconnection of the presenting connection.
<code>swf_verification_unsupported_rejects</code>	A counter of the number of SWF verification unsupported rejections. When a version of Flash Player that doesn't support SWF verification connects to an application that requires SWF verification, the unsupported rejection count is increased and the connecting client is disconnected.
<code>swf_verification_matches</code>	A counter of the total number of matches. When an authentic SWF verification credential is presented, this number increases. There may be more than one match per connection.
<code>swf_verification_remote_misses</code>	A counter of the proxy/remote server process's missed SWF verification attempts. Whenever a proxy/remote server receives a SWF verification attempt it looks to its local cache for valid SWF verification. If it does not locate a match it logs a remote miss and defers to the origin to answer the verification attempt.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a String describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The `timestamp` response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## getLicenseInfo()

**RTMP/E** `getLicenseInfo()` : Object

**HTTP** `http://www.example.com:1111/admin/getLicenseInfo?auser=username&apswd=password`

Retrieves complete license information including information on the maximum bandwidth and maximum number of connections, adaptors, virtual hosts, and CPUs that are allowed by the license. License information for all your licenses is summarized and then followed by specific information about each license. This call returns information about license keys (also called license numbers) and license files.

## Availability

Flash Communication Server 1.0.

## Parameters

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

## Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object. The data object has the same properties as the data element in the XML returned by an HTTP call.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure (this XML is for a license file):

**Note:** The value `-1` means unlimited.

```

<result>
  <level>status</level>
  <code>NetConnection.Call.Success</code>
  <timestamp>10/31/2007 1:36:05 PM</timestamp>
  <data>
    <name>Flash Media Interactive Server</name>
    <version>3.0.0</version>
    <build>d1107</build>
    <copyright>Copyright (c) 1993-2007 Adobe Systems Incorporated.
All Rights Reserved.</copyright>
    <max_connections>unlimited</max_connections>
    <max_edge_connections>unlimited</max_edge_connections>
    <max_adaptors>3</max_adaptors>
    <max_vhosts>unlimited</max_vhosts>
    <max_cpu>-1</max_cpu>
    <max_bandwidth>unlimited</max_bandwidth>
    <max_core_process>-1</max_core_process>
    <proxy>enabled</proxy>
    <custom_plugin>enabled</custom_plugin>
    <license_stacking>yes</license_stacking>
    <apps>enabled</apps>
    <web-proxy>enabled</web-proxy>
    <screenshare>enabled</screenshare>
    <app-isolation>enabled</app-isolation>
    <inst-isolation>enabled</inst-isolation>
    <license_files>
      <_0>
        <filename>license0108.lic</filename>
        <description>Beta version</description>
        <type>FlashMediaServer</type>
        <id></id>
        <account-id></account-id>
        <expires>Jan 01, 2008</expires>
        <limits>
          <connections>
            <value>unlimited</value>
          </connections>
          <bandwidth>

```

```

        <value>unlimited</value>
      </bandwidth>
    <vhosts>
      <value>unlimited</value>
    </vhosts>
  </limits>
  <features>
    <apps>
      <value>enabled</value>
    </apps>
    <proxy>
      <value>enabled</value>
    </proxy>
    <screenshare>
      <value>enabled</value>
    </screenshare>
    <web-proxy>
      <value>enabled</value>
    </web-proxy>
    <app-isolation>
      <value>enabled</value>
    </app-isolation>
    <inst-isolation>
      <value>enabled</value>
    </inst-isolation>
  </features>
</_0>
</license_files>
<active_profile>N/A</active_profile>
<key_details></key_details>
</data>
</result>

```

If the call succeeds, it returns XML with the following structure (this XML is for a license key):

```

<result>
  <level>status</level>
  <code>NetConnection.Call.Success</code>
  <timestamp>10/31/2007 1:34:17 PM</timestamp>
  <data>
    <name>Flash Media Interactive Server</name>
    <version>3.0.0</version>
    <build>d1107</build>
    <copyright>Copyright (c) 1993-2007 Adobe Systems Incorporated.
All Rights Reserved.</copyright>
    <max_connections>unlimited</max_connections>
    <max_edge_connections>unlimited</max_edge_connections>
    <max_adaptors>3</max_adaptors>
    <max_vhosts>unlimited</max_vhosts>
    <max_cpu>4</max_cpu>
    <max_bandwidth>unlimited</max_bandwidth>
    <max_core_process>-1</max_core_process>
    <proxy>enabled</proxy>
    <custom_plugin>enabled</custom_plugin>
    <license_stacking>yes</license_stacking>
    <apps>enabled</apps>
    <web-proxy>enabled</web-proxy>
    <app-isolation>enabled</app-isolation>
    <inst-isolation>enabled</inst-isolation>
    <license_files>
  </license_files>

```

```

<active_profile>N/A</active_profile>
<key_details>
  <_0>
    <key></key>
    <product_code>N/A</product_code>
    <type>N/A</type>
    <family>N/A</family>
    <edition>Flash Media Interactive Server</edition>
    <edition_text>N/A</edition_text>
    <max_connections>unlimited</max_connections>
    <max_adaptors>3</max_adaptors>
    <max_vhosts>unlimited</max_vhosts>
    <max_cpu>4</max_cpu>
    <max_bandwidth>unlimited</max_bandwidth>
    <expires></expires>
    <valid>true</valid>
  </_0>
  <_1>
    <key></key>
    <product_code>N/A</product_code>
    <type>N/A</type>
    <family>N/A</family>
    <edition>Flash Media Streaming Server</edition>
    <edition_text>N/A</edition_text>
    <max_connections>unlimited</max_connections>
    <max_adaptors>1</max_adaptors>
    <max_vhosts>unlimited</max_vhosts>
    <max_cpu>4</max_cpu>
    <max_bandwidth>unlimited</max_bandwidth>
    <expires></expires>
    <valid>true</valid>
  </_1>
</key_details>
</data>
</result>

```

If the call fails, it returns XML with the following structure:

```

<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>

```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## getLiveStreams()

**RTMP/E** getLiveStreams(appInst:String) : Object

**HTTP** http://www.example.com:1111/admin/getLiveStreams?auser=username&apswd=password  
&appInst=name

Gets an array of strings that contains the names of all the live streams that are currently publishing to the specified instance of an application.



## Availability

Flash Communication Server 1.0.

## Parameters

**appInst** A String indicating the name of the instance of the application, in the form `application_name/instance_name`.

**ouser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

## Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an array of strings containing the names of all the live streams that are currently publishing to the specified instance of an application.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The *timestamp* response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## See also

[getLiveStreamStats\(\)](#)

# getLiveStreamStats()

**RTMP/E** `getLiveStreamStats(appInst:String, stream:String) : Object`

**HTTP** `http://www.example.com:1111/admin/getLiveStreamStats?ouser=username&apswd=password&appInst=name&stream=name`

Returns detailed information about a live stream.

## Availability

Flash Communication Server 1.0.

## Parameters

**appInst** A String indicating the name of the instance of the application, in the form `application_name/instance_name`.

**stream** A String indicating the name of the stream.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

## Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object. The data object has the following properties:

Property	Description
<code>publisher</code>	Object; publisher statistics. The object has the following properties:  <code>name</code> : String; the name of the published live stream.  <code>time</code> : Date object; time that the stream was published. This property is a duplicate of <code>publish_time</code> and exists for backward compatibility.  <code>type</code> : String; the type of stream for the publisher. The value is "publishing".  <code>client</code> : Number; the client ID of the publisher.  <code>stream_id</code> : Number; the stream ID of the publisher.  <code>publish_time</code> : Date object; time that the stream was published.  <code>client_type</code> : String; the string type of the publishing client.
<code>subscribers</code>	Array of subscriber statistics. The array contains a <code>subscriber</code> property that is an object containing the following properties:  <code>client</code> : Number; user ID.  <code>subscribe_time</code> : Date object; the time that the user subscribed to the stream.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the data object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
```

```
<description></description>
<timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

#### See also

[getLiveStreams\(\)](#)

## getMsgCacheStats()

**RTMP/E** `getMsgCacheStats()` : Object

**HTTP** `http://www.example.com:1111/admin/getMsgCacheStats?auser=username&apswd=password`

Returns server TCMessages cache statistics. You must be a server administrator to perform this command.

#### Availability

Flash Communication Server 1.0.

#### Parameters

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

#### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object. The data object has the following message cache statistics:

Property	Description
<code>allocated</code>	Number; total number of message objects allocated.
<code>reused</code>	Number; total number of objects used.
<code>size</code>	Number; current size.

Property	Description
thread_count	Total number of per-thread pools in use.
units	<p>Object with the following properties:</p> <p>global_size    The size of the global pool that is available.</p> <p>thread_size    The total size of the per-thread pool that is available.</p> <p>size    The total size of the global and per-thread pool that is available.</p> <p>reused    The total number of messages reused.</p> <p>allocated    The total number of messages allocated from the heap.</p> <p>released    The total number of messages released back to the heap.</p> <p>relocated    The total number of messages that have been reallocated.</p> <p>bulk_allocated    The total number of messages allocated from the global pool.</p> <p>bulk_released    The total number of messages released back to the global pool.</p> <p>huge_allocated    The total number of huge messages (greater than 16 kilobytes) allocated.</p> <p>huge_released    The total number of huge messages (greater than 16 kilobytes) released.</p>
bytes	<p>Object with the following properties:</p> <p>global_size    The size of the global pool that is available.</p> <p>thread_size    The total size of the per-thread pool that is available.</p> <p>size_total    The size of the global and per-thread pool that is available.</p> <p>reused    The total number of bytes reused.</p> <p>allocated    The total number of bytes allocated from the heap.</p> <p>released    The total number of bytes released back to the heap.</p> <p>relocated    The total number of bytes that have been reallocated.</p> <p>bulk_allocated    The total number of bytes allocated from the global pool.</p> <p>bulk_released    The total number of bytes released back to the global pool.</p> <p>huge_allocated    The total number huge messages (greater than 16 kilobytes) allocated in bytes.</p> <p>huge_released    The total number of huge messages (greater than 16 kilobytes) released in bytes.</p>

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the data element are elements for each property of the data object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## getNetStreams()

**RTMP/E** `getNetStreams(appInst:String) : Object`

**HTTP** `http://www.example.com:1111/admin/getNetStreams?auser=username&apswd=password&appInst=name`

Returns an array of numbers that represent the server-assigned IDs of all the network streams that are currently connected to the specified instance of the application.

### Availability

Flash Communication Server 1.0.

### Parameters

**appInst** A String indicating the name of the instance of the application, in the form `application_name/instance_name`.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an array of numbers. The numbers represent the server-assigned IDs of all network streams that are currently connected to the specified instance of the application.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

#### See also

[getNetStreamStats\(\)](#)

## getNetStreamStats()

**RTMP/E** `getNetStreamStats(appInst:String, streamids:Number) : Object`

**HTTP** `http://www.example.com:1111/admin/getNetStreamStats?auser=username&apswd=password&appInst=name&streamids=number`

Gets detailed information for one or more network streams that are connecting to the specified instance of an application.

#### Availability

Flash Communication Server 1.0.

#### Parameters

**appInst** A String indicating the name of the instance of the application, in the form *application\_name/instance\_name*.

**streamids** A Number indicating the ID of the network stream or an array of numbers that represents the network stream ID. To get information for all the network streams that are currently connected, specify a value of -1 for the *streamids* parameter.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

#### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a *level* property of *status*, a *code* property of *NetConnection.Call.Success*, and a *data* property that is an array of statistics for the network stream. Each element in the array is an object that has the following properties:

Property	Name
<i>streamid</i>	Number; stream ID.
<i>name</i>	String; stream name or empty if the stream is idle.

Property	Name
type	String; stream type. Possible values are shown in the following list: "idle" "publishing" "playing live" "play recorded"
client	Number; user ID.
time	ActionScript Date object; possible values are shown in the following list: If type = idle, value is 0. If type = publishing, value is the time the stream was published. If type = playing live, value is the time the playback of the stream started. If type = play recorded, value is the time the playback of the stream started.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The `timestamp` response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

**See also**

[getNetStreams\(\)](#)

## getRecordedStreams()

**RTMP/E** `getRecordedStreams(appInst:String)` : Object

**HTTP** `http://www.example.com:1111/admin/getRecordedStreams?auser=username&apswd=password&appInst=name`

Returns an array containing the names of all the recorded streams currently playing from a particular instance of an application.

### Availability

Flash Media Server 2.0.

### Parameters

**appInst** A String indicating the name of the application or instance of the application, in the form `application_name[/instance_name]`.

**ouser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an array of all the recorded stream names. The extended name of a recorded stream is used. The name is encoded with `keyName*type:streamName`, where `keyName` is the virtual key, `type` is the stream type (for example: `flv`, `mp3`, and so on), and `streamName` is the text name of the stream.

The properties are defined in the following table:

Property	Name
<code>streamName</code>	String; name of the recorded stream.
<code>type</code>	String; type of the recorded stream.
<code>keyName</code>	String; virtual key of this recorded stream.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The *timestamp* response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

### See also

[getRecordedStreamStats\(\)](#)



## getRecordedStreamStats()

**RTMP/E** `getRecordedStreamStats(appInst:String, stream:String) : Object`

**HTTP** `http://www.example.com:1111/admin/getRecordedStreamStats?auser=username&apswd=password&appInst=name&stream=name`

Returns detailed information about a recorded stream.

### Availability

Flash Media Server 2.0.

### Parameters

**appInst** A String indicating the name of the instance of the application, in the form `application_name/instance_name`.

**stream** A String indicating the name of the stream. If the stream has a nondefault virtual key or type, these items should be encoded into the stream name. The following is an example of how to encode key and type:

`key?type:name` and `on2key?flv:myStream`.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object that contains the following properties:

Property	Name
<code>cache_bytes</code>	Number; total number of cached bytes of this recorded stream.
<code>cache_segments</code>	Number; number of segments cached for this recorded stream.
<code>cache_hits</code>	Number; number of hits on this stream within the cache.
<code>cache_misses</code>	Number; number of misses on the cache for this stream.
<code>modified_time</code>	ActionScript Date object; date when this file was last modified.
<code>size</code>	Number; number of bytes in this recorded file.
<code>length</code>	Number; length of this file in seconds.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
```

```

<code></code>
<description></description>
<timestamp></timestamp>
</result>

```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

#### See also

[getRecordedStreams\(\)](#)

## getScriptStats()

**RTMP/E** `getScriptStats(appInst:String) : Object`

**HTTP** `http://www.example.com:1111/admin/getScriptStats?auser=username&apswd=password&appInst=name`

Gets the performance data for a script running on the specified instance of an application.

#### Availability

Flash Communication Server 1.0.

#### Parameters

**appInst** A String indicating the name of the instance of the application, in the form `application_name/instance_name`.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

#### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object. The data object has the following properties:

Property	Description
<code>time_high_water_mark</code>	Number; maximum amount of time, in seconds, the script has taken to execute an event.
<code>queue_size</code>	Number; total number of events currently in the script engine queue.
<code>total_processed</code>	Number; total number of events processed by the script engine.
<code>total_process_time</code>	Number; number of seconds taken to process the number of events in <code>total_processed</code> .
<code>queue_high_water_mark</code>	Number; maximum number of events in the queue.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The `timestamp` response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## getServerStats()

**RTMP/E** `getServerStats()` : Object

**HTTP** `http://www.example.com:1111/admin/getServerStats?auser=username&apswd=password`

Retrieves the server status and statistics about the operation of the server, including the length of time the server has been running and I/O and message cache statistics.

You must be a server administrator to perform this operation.

If you only need information about the I/O characteristics of the server, use the `getIOStats()` method instead.

### Availability

Flash Communication Server 1.0.

### Parameters

`auser` A String indicating the user name of the administrator.

`apswd` A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object. The `data` object has the following properties:

Property	Description
launchTime	ActionScript Date object; time the server was started.
up_time	Number; length of time, in seconds, that the server has been running.
io	I/O statistics, returned as an object with the following properties:  msg_in: Number; total number of messages processed by the server.  msg_out: Number; total number of messages sent by the server.  bytes_in: Number; total number of bytes read by the server.  bytes_out: Number; total number of bytes written by the server.  reads: Number; total number of system read calls.  writes: Number; total number of system writes.  connected: Number; total number of active socket connections to the server.  total_connects: Number; total number of socket connections to the server.  total_disconnects: Number; total number of socket disconnections from the server.
msg_cache	Flash Media Server message packet cache statistics, returned as an object with the following properties:  allocated: Number; total number of message objects allocated.  reused: Number; total number of objects reused.  size: Number; size of the cache, in number of message packets.
memory_Usage	Number. On Microsoft Windows NT 4.0, the approximate percentage of the last 1000 pages of physical memory in use.  On Windows 2000 or Windows XP, the approximate percentage of total physical memory in use.
cpu_Usage	Number; approximate percentage of CPU in use by the Flash Media Server processes—not by the entire system.
swf_verification_attempts	A counter of the number of SWF verification attempts made. Represents the total SWF verification credentials passed to the server for checking. There may be more than one credential presented per connection.
swf_verification_exceptions	A counter of the number of SWF verification exceptions made. Exceptions are allowed through explicit configuration in the Application.xml file that allows certain user agents to bypass the requirement for SWF verification. Every connection allowed as an exception is counted here.
swf_verification_failures	A counter of the number of SWF verification failures. Failures result from the presentation of SWF verification credentials that are found not to be a match for any loaded credential. Each failure corresponds to a disconnection of the presenting connection.
swf_verification_unsupported_rejects	A counter of the number of SWF verification unsupported rejections. When a version of Flash Player that doesn't support SWF verification connects to an application that requires SWF verification, the unsupported rejection count is increased and the connecting client is disconnected.
swf_verification_matches	A counter of the total number of matches. When an authentic SWF verification credential is presented, this number increases. There may be more than one match per connection.
swf_verification_remote_misses	A counter of the proxy/remote server process's missed SWF verification attempts. Whenever a proxy/remote server receives a SWF verification attempt it looks to its local cache for valid SWF verification. If it does not locate a match it logs a remote miss and defers to the origin to answer the verification attempt.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The *timestamp* response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

#### See also

[getIOStats\(\)](#)

## getSharedObjects()

**RTMP/E** `getSharedObjects(appInst:String) : Object`

**HTTP** `http://www.example.com:1111/admin/getSharedObjects?auser=username&apswd=password&appInst=name`

Gets the names of all the shared objects that are currently active.

#### Availability

Flash Communication Server 1.0.

#### Parameters

**appInst** A String indicating the name of the instance of the application, in the form `application_name/instance_name`.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

#### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object. The data object has the following properties:

Property	Description
<code>persistent</code>	Array of strings that contain persistent shared object names.
<code>volatile</code>	Array of strings that contain shared object names that are not persistent.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

This call fails if you supply an application or instance name that does not exist.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

#### See also

[getSharedObjectStats\(\)](#)

## getSharedObjectStats()

**RTMP/E** `getSharedObjectStats(appInst:String, sharedObject:String, persistent:Boolean) :`  
Object

**HTTP** `http://www.example.com:1111/admin/getSharedObjectStats?auser=username&apswd=password`  
&appInst=name&sharedObject=name&persistent=value

Gets detailed information about a shared object.

#### Availability

Flash Communication Server 1.0.

#### Parameters

**appInst** A String indicating the name of the instance of the application, in the form `application_name/instance_name`.

**sharedObject** A String indicating the name of the shared object. You can get the names of all active shared objects by using the `getSharedObjects()` method.

**persistent** A Boolean value: `true` (1) for persistent; `false` (0) for nonpersistent.

**auser** A String indicating the user name of the administrator.

`apswd` A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object. The `data` object has the following properties:

Property	Description
<code>version</code>	Number; version number of the shared object.
<code>num_properties</code>	Number; total number of properties in the shared object.
<code>msg_in</code>	Number; total messages in to the shared object.
<code>msg_out</code>	Number; total messages out from the shared object.
<code>total_connects</code>	Number; total number of connections to the shared object.
<code>total_disconnects</code>	Number; total number of disconnections from the shared object.
<code>connected</code>	Number; number of active subscribers.
<code>resync_depth</code>	Number; maximum version retained before resynchronization. If the difference between the server version number and the client version number is greater than the <code>resync_depth</code> value, Flash Media Server sends only changes between versions.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The *timestamp* response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

### See also

[getSharedObjects\(\)](#)

## getUsers()

**RTMP/E** `getUsers(appInst:String) : Object`

**HTTP** `http://www.example.com:1111/admin/getUsers?auser=username&apswd=password&appInst=name`

Gets an array of strings that represent the server-assigned IDs of all users who are connected to the specified instance of an application.

### Availability

Flash Communication Server 1.0.

### Parameters

**appInst** A String indicating the name of the instance of the application, in the form `application_name/instance_name`.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an array of strings that represent the IDs of users who are connected to the specified instance of an application. The user IDs are assigned by the server.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data>
    <names></names>
  </data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The *timestamp* response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

### See also

[getUserStats\(\)](#)



## getUserStats()

**RTMP/E** `getUserStats(appInst:String, userid:String) : Object`

**HTTP** `http://www.example.com:1111/admin/getUserStats?auser=username&apswd=password  
&appInst=name&userid=ID`

Gets the performance data for the connection of a specified user.

### Availability

Flash Communication Server 1.0.

### Parameters

**appInst** A String indicating the name of the instance of the application, in the form `application_name/instance_name`.

**userid** A String indicating the user ID, as assigned by the server. You can retrieve a user ID with the `getUsers()` method.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object. The data object has the following properties:

Property	Description
<code>connect_time</code>	ActionScript Date object; time, in seconds, that the user has been connected to the specified instance of the application.
<code>msg_in</code>	Number; total number of messages processed by this user.
<code>msg_out</code>	Number; total number of messages sent by this user.
<code>msg_dropped</code>	Number; total number of messages dropped by this user.
<code>bytes_in</code>	Number; total number of bytes read by this user.
<code>bytes_out</code>	Number; total number of bytes written by this user.
<code>msg_queue</code>	Object that contains the client message queue statistics. The <code>msg_queue</code> object contains the following properties:  <code>total_queues</code> : Total number of queues for this client.  <code>audio</code> : Total number of audio messages in all audio queues.  <code>video</code> : Total number of video messages in all video queues.  <code>other</code> : Total number of command/data messages in the other queue.
<code>protocol</code>	String; protocol used by the client to connect to the server ( <code>rtmp</code> , <code>rtmpe</code> , <code>rtmpt</code> , or <code>rtmps</code> .)
<code>stream_ids</code>	Array of numbers that represent the stream IDs.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The *timestamp* response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

#### See also

[getUsers\(\)](#)

## getVHosts()

**RTMP/E** `getVHosts([adaptor:String]) : Object`

**HTTP** `http://www.example.com:1111/admin/getVHosts?auser=username&apswd=password`  
[&adaptor=name]

Returns an array of virtual hosts defined for the specified adaptor. You must be a server administrator to call this method.

#### Availability

Flash Communication Server 1.0.

#### Parameters

**adaptor** A String indicating the user name of the adaptor. If not specified, it is assumed to be `"_defaultRoot_"`.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

#### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an array of strings containing the names of all the virtual hosts.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the data element are elements for each property of the data object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## getVHostStats()

**RTMP/E** getVHostStats([adaptor:String, vhost:String]) : Object

**HTTP** http://www.example.com:1111/admin/getVHostStats?adaptor=username&apswd=password  
&adaptor=name&vhost=name

Returns aggregate performance data for all instances for all applications for the specified virtual host. You must be a server administrator to call this method.

### Availability

Flash Communication Server 1.0.

### Parameters

**adaptor** A String indicating the user name of the adaptor. If not specified, it is assumed to be `_defaultRoot_`.

**vhost** A String indicating the user name of the virtual host. If not specified, it is assumed to be `_defaultVHost_`.

**adaptor** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `data` property that is an object containing the virtual host performance data. The data object has the following properties:

Property	Description
msg_in	Number; total number of messages processed by this virtual host.
msg_out	Number; total number of messages sent by this virtual host.
msg_dropped	Number; total number of messages dropped by this virtual host.
bytes_in	Number; total number of bytes read by this virtual host.
bytes_out	Number; total number of bytes written by this virtual host.
accepted	Number; total number connections accepted by this virtual host.
rejected	Number; total number of connections rejected by this virtual host.
connected	Number; total number of connections currently active.
total_apps	Number; total number of applications that have been created.
total_connects	Number; total number of connections to the server.
total_disconnects	Number; total number of disconnections from the server.
total_instances_loaded	Number; total number of instances that have been loaded.  This property does not represent the total number of active instances loaded. To get the number of active instances loaded, subtract the value of <code>total_instances_unloaded</code> from <code>total_instances_loaded</code> .
total_instances_unloaded	Number; total number of instances that have been unloaded.
bw_in	Number; current bandwidth in, in bytes per second.
bw_out	Number; current bandwidth out, in bytes per second.
normal_connects	Number; total number of normal connections.
virtual_connects	Number; total number of connections through a remote edge.
group_connects	Number; total number of remote edges that are connected.
service_connects	Number; total number of service connections.
service_requests	Number; total number of services requested.
admin_connects	Number; total number of administrator connections.
debug_connects	Number; total number of debug connections.
swf_verification_attempts	A counter of the number of SWF verification attempts made. Represents the total SWF verification credentials passed to the server for checking. There may be more than one credential presented per connection.
swf_verification_exceptions	A counter of the number of SWF verification exceptions made. Exceptions are allowed through explicit configuration in the <code>Application.xml</code> file that allows certain user agents to bypass the requirement for SWF verification. Every connection allowed as an exception is counted here.
swf_verification_failures	A counter of the number of SWF verification failures. Failures result from the presentation of SWF verification credentials that are found not to be a match for any loaded credential. Each failure corresponds to a disconnection of the presenting connection.

Property	Description
<code>swf_verification_unsupported_rejects</code>	A counter of the number of SWF verification unsupported rejections. When a version of Flash Player that doesn't support SWF verification connects to an application that requires SWF verification, the unsupported rejection count is increased and the connecting client is disconnected.
<code>swf_verification_matches</code>	A counter of the total number of matches. When an authentic SWF verification credential is presented, this number increases. There may be more than one match per connection.
<code>swf_verification_remote_misses</code>	A counter of the proxy/remote server process's missed SWF verification attempts. Whenever a proxy/remote server receives a SWF verification attempt it looks to its local cache for valid SWF verification. If it does not locate a match it logs a remote miss and defers to the origin to answer the verification attempt.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The `timestamp` response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## ping()

**RTMP/E** `ping()` : Object

**HTTP** `http://www.example.com:1111/admin/ping?auser=username&apswd=password`

Verifies that the server is running; the server responds with a status message. You can use this method to check the status of the server.

### Availability

Flash Communication Server 1.0.

### Parameters

**auser** A String indicating the user name of the administrator.

`apswd` A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, a `code` property of `NetConnection.Call.Success`, and a `timestamp` property that is a `Date` object. The `Date` object indicates the time that the method was executed.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Call.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## reloadApp()

**RTMP/E** `reloadApp(appInst:String) : Object`

**HTTP** `http://www.example.com:1111/admin/reloadApp?user=username&apswd=password&appInst=name`

Shuts down an instance of the application, if running, and reloads it. All users are disconnected.

**Note:** If you specify an application name, this call reloads the `_definst_` instance.

After this method executes, users must reconnect to the application or instance of the application.

Call this method to preload an instance of an application or reload it when application configuration changes are made or the script that is associated with the application has been changed.

### Availability

Flash Communication Server 1.0.

**Parameters**

**appInst** A String indicating the name of the application or instance of the application, in the form `application_name[/instance_name]`. To reload the default instance of an application, specify only the name of the application.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

**Returns**

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status` and a `code` property of `NetConnection.Call.Success`.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Admin.Command.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

Nested in the `data` element are elements for each property of the `data` object listed in the RTMP/E section.

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

**See also**

[unloadApp\(\)](#)

## removeAdmin()

**RTMP/E** `removeAdmin(user:String [,scope:String]) : Object`

**HTTP** `http://www.example.com:1111/admin/removeAdmin?auser=username&apswd=password&user=name [&scope=scope]`

Removes an administrator from the system. Depending on the parameters you specify, you can remove server administrators or virtual host administrators.

You must be a server administrator to remove an administrator from the system.

## Availability

Flash Communication Server 1.0.

## Parameters

**user** A String indicating the user name of the administrator being removed.

**scope** A String indicating which administrator you want to remove.

To remove a virtual host administrator from the virtual host to which you're connected, omit this parameter. To remove a virtual host administrator from a different virtual host, specify the virtual host as `adaptor_name/virtual_hostname`.

To remove a server administrator, specify `server`.

**ouser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

## Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status` and a `code` property of `NetConnection.Call.Success`.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Admin.Command.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

If the specified administrator does not exist, the call fails.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
</result>
```

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The *timestamp* response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## Example

The following examples show the `removeAdmin()` method:

```
nc_admin = new NetConnection();
nc_admin.connect("rtmp://localhost:1111/admin", "JLee", "x52z49ab");

/* Remove a server administrator named "DYoung" */
nc-admin.call("removeAdmin", new onRemoveAdmin(), "DYoung", "server");

/* Remove a virtual host administrator named "LPark" */
```



```
nc_admin.call("removeAdmin", new onRemoveAdmin(), "LPark");

/* Remove a virtual host administrator "JGarcia" from vhost tree.oak.com */
nc_admin.call("removeAdmin", new onRemoveAdmin(), "JGarcia", "_defaultRoot/" +
tree.oak.com");
```

**See also**[addAdmin\(\)](#)

## removeApp()

**RTMP/E** removeApp(appName:String) : Object**HTTP** http://www.example.com:1111/admin/removeApp?user=username&apswd=password&appName=name

Removes the specified application or instance of an application from the virtual host. First, all instances of the specified application are unloaded. Then the application directory is removed from the virtual host. If you specify an instance of an application, only that instance is unloaded and removed, and all streams and shared objects for that instance are deleted.

**Availability**

Flash Communication Server 1.0.

**Parameters**

**appName** A String indicating the name of the application or instance of the application you want to remove, in the form application\_name[/instance\_name].

**user** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

**Returns**

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status` and a `code` property of `NetConnection.Call.Success`.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Admin.Command.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
</result>
```

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

### Example

The following example shows how you remove the entire application ChatApp and then shows how you remove a specific instance Instance1 of an application ChatApp.

```
nc_admin = new NetConnection();
nc_admin.connect("rtmp://localhost:1111/admin", "JGarcia", "ezcabby1");

/* Removes the application along with all instances of the application */

nc_admin.call("removeApp", new onRemoveApp(), "ChatApp");

/* Removes only the specified instance */

nc_admin.call("removeApp", new onRemoveApp(), "ChatApp/Instance1");
```

### See also

[addApp\(\)](#)

## removeVHostAlias()

**RTMP/E** removeVHostAlias(vhost:String, alias:String) : Object

**HTTP** <http://www.example.com:1111/admin/removeVHostAlias?auser=username&apswd=password&vhost=name&alias=name>

Removes an alias from a virtual host. Aliases are alternative names for virtual hosts that are used as targets by incoming Flash Media Server connections. When you removing an alias, that name is no longer available for incoming connections.

### Availability

Flash Media Server 2.0.

### Parameters

- vhost** A String indicating the virtual host from which to remove an alias.
- alias** A String indicating the alias name to remove from the specified virtual host.
- auser** A String indicating the user name of the administrator.
- apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status`, and a `code` property of `NetConnection.Call.Success`.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
```

```
</result>
```

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

### Example

```
nc_admin = new NetConnection();
nc_admin.connect("rtmp://localhost:1111/admin", "JLee", "x52z49ab")
admin_nc.call("removeVHostAlias", new Result(), "myvhost", "myalias");
```

### See also

[addVHostAlias\(\)](#)

## restartVHost()

**RTMP/E** restartVHost([vhost:String]) : Object

**HTTP** [http://www.example.com:1111/admin/restartVHost?auser=username&apswd=password \[&vhost=vhost\]](http://www.example.com:1111/admin/restartVHost?auser=username&apswd=password [&vhost=vhost])

Restarts a virtual host that is currently running. Restarting a virtual host disconnects all connected users, unloads all currently loaded applications, and reloads the configuration files for that virtual host. For restartVHost() to work properly, there must be at least one application instantiated on the virtual host.

If you make changes to the configuration files for a virtual host, you can call this method to restart the virtual host without restarting the server.

Users must reconnect each time you restart a virtual host. Before calling this method, you might want to take steps to notify connected users.

Virtual host administrators can only restart the virtual hosts to which they are connected. You must be a server administrator to start a virtual host other than the one to which you're connected.

**Note:** To start a virtual host that is stopped, call the startVHost() method. You must be a server administrator to call this method.

### Availability

Flash Communication Server 1.0.

### Parameters

**vhost** A String indicating which virtual host you want to restart.

To restart the virtual host to which you're connected, omit this parameter.

To restart a different virtual host, specify the virtual host as adaptor\_name/virtual\_hostname.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status` and a `code` property of `NetConnection.Call.Success`.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Admin.Command.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
</result>
```

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The *timestamp* response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

### Example

The following example shows a call to restart the connected virtual host and then a call to restart a specified virtual host `tree.oak.com` on the default adaptor:

```
nc_admin = new NetConnection();
nc_admin.connect("rtmp://localhost:1111/admin", "JLee", "x52z49ab");

/* Restart the connected virtual host */
nc_admin.call("restartVHost", new onRestartVHost());

/* Restart the virtual host tree.oak.com on the default adaptor. */
nc_admin.call("restartVHost", new onRestartVHost(), "_defaultRoot_/tree.oak.com");
```

### See also

[reloadApp\(\)](#), [startVHost\(\)](#), [stopVHost\(\)](#)

## setConfig()

**RTMP/E** `setConfig(key:String, value:String, [,scope:String]) : Object`

**HTTP** `http://www.example.com:1111/admin/setConfig?auser=username&apswd=password&key=string&value=string [&scope=scope]`

This API has been deprecated; use `setConfig2()` instead. Changes the value of the specified configuration key in a specified configuration file. For a description of the XML configuration files, see *Adobe Flash Media Server Configuration and Administration Guide*.

Virtual host administrators can change the values of configuration keys in the `Vhost.xml` file and `Application.xml` files for their own virtual hosts. You must be a server administrator to change the values of most configuration keys in the `Server.xml` and `Adaptor.xml` files.

**Note:** It is possible to have more than one XML tag with the same name at the same level in the XML tree. In the configuration file, you should distinguish such tags by using a name attribute in the XML tag (for example, if you have more than one virtual host: `<VirtualHost name="www.redpin.com"></VirtualHost>`.) When you call the `setConfig()` method and specify the configuration subkeys, you can indicate which tag you want by specifying the tag name, followed by a colon and the correct name attribute, for example:

`Adaptor:_defaultRoot_/VirtualHost:www.redpin.com`.

### Availability

Flash Communication Server 1.0.

### Parameters

**key** A String indicating the configuration key for which you want to change the value.

A key is specified as a list of subkeys delimited by slashes (/). The first subkey specifies the XML configuration file that contains the desired configuration key. Subsequent subkeys correspond to tags that are relative to that XML configuration file; the hierarchy and names of the subkeys match the tags in the XML file.

Depending on your permissions, you can change the values of configuration keys for the following files:

- For the `Server.xml` file, specify `Admin` or `Server` as the first subkey. All subsequent keys correspond to tags that are relative to the `<Admin>` or `<Server>` tag in the `Server.xml` file.

You must be a server administrator to set configuration keys in the `<Server>` tag.

Virtual host administrators can set configuration keys for their own virtual host only. They might not be able to set certain kinds of sensitive information; for example, they can set their own password, but they cannot set other virtual host administrators' passwords or permission settings.

- For the `Adaptor.xml` file, specify as the first subkey `Adaptor:adaptor_name`, where `adaptor_name` is the name of the adaptor. All subsequent keys correspond to tags relative to the `<Adaptor>` tag in the `Adaptor.xml` file.
- For the `Vhost.xml` file, specify as the first subkey `Adaptor:adaptor_name/VirtualHost:vhost_name`, where `vhost_name` is the name of the virtual host. All subsequent keys correspond to tags relative to the `<VirtualHost>` tag in the `Vhost.xml` file.
- For the `Application.xml` file of an application that is running on the same virtual host to which you connected when you logged on to the administration server, specify as the first subkey `Application:app_name`, where `app_name` is the name of the application.

To get a key in the `Application.xml` file for an application that is running on a different virtual host, specify the full key: `Adaptor:adaptor_name/VirtualHost:vhost_name/Application:app_name`. You must also specify the `scope` parameter; see the code example later in this entry.

To get the default `Application.xml` file, specify `Application` without the colon (:) and the `app_name` attribute.

**Note:** If a subkey is specified but a corresponding tag doesn't exist in the XML file, a new tag is created in the XML file.

**value** A String indicating the value to set for the specified configuration key.

**scope** A String; to change a configuration key in the Server.xml file, Adaptor.xml file, or Vhost.xml files, you must specify a slash (/) for this parameter.

To change a configuration key in the Application.xml file for an application that is running on the same virtual host to which you connected when you logged on to Flash Media Server, omit this parameter.

**Note:** To determine the adaptor or virtual host to which you're connected, use the `getAdminContext()` method.

**ouser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

## Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of status and a `code` property of `NetConnection.Call.Success`. The value of the configuration key is changed.

If the call fails, the server sends a reply information object with a `level` property of error, a `code` property of `NetConnection.Call.BadValue`, and a `description` property that contains a string describing the cause of the failure.

This call fails if the specified configuration key cannot be found or if you do not have permissions to change its value.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
</result>
```

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## Examples

The following examples show how to set new values for configuration keys in each of the four XML files:

```
// Establish connection to server
nc_admin = new NetConnection();
nc_admin.connect("rtmp://localhost:1111/admin", "LGreen", "123jn098");

// For a virtual host administrator, change key in Server.xml
// Set Password for user LGreen to "strawman28"
key = "Admin/Server/UserList/User:LGreen/Password";
val = "strawman28"
nc_admin.call("setConfig", new onSetConfig(), key, val, "/");

// For a server administrator, change key in Server.xml
// Set LicenseInfo to "Helloworld"
key = "Server/LicenseInfo";
val = "Helloworld";
```

```
nc_admin.call("setConfig", new onSetConfig(), key, val, "/");

// Change key in Adaptor.xml
// Set HostPort to 128.0.0.1:1938
key = "Adaptor:_defaultRoot_/HostPortList/HostPort";
val = "128.0.0.1:1938";
nc_admin.call("setConfig", new onSetConfig(), key, val, "/");

// Change key in Vhost.xml
// Set RecordAccessLog to true
key = "Adaptor:_defaultRoot_/VirtualHost:_defaultVHost_/RecordAccessLog";
val = "true";
nc_admin.call("setConfig", new onSetConfig(), key, val, "/");

// Change key in Application.xml for an application on the virtual host
// you connected to when you logged on to the administration server.
// Note that the previous subkeys and the final parameter "/" are not
// necessary.
key = "Application:FinanceApp/RecordAppLog";
val = "true";
nc_admin.call("setConfig", new onSetConfig(), key, val);

// Change key in Application.xml for a different virtual host.
// The adaptor and virtual host names and the second parameter are necessary.
key = "Adaptor:_defaultRoot_/VirtualHost:www.redpin.com/Application:ChatApp/RecordAppLog";
val = "true";
nc_admin.call("setConfig", new onSetConfig(), key, val, "/");
```

**See also**

[getAdminContext\(\)](#), [getConfig\(\)](#)

## setConfig2()

**RTMP/E** setConfig2(key:String, value:String, scope:String) : Object

**HTTP** <http://www.example.com:1111/admin/setConfig2?user=username&passwd=password&key=string&value=string&scope=scope>

Changes the value of the specified configuration key in a specified configuration file. Flash Media Server has six server configuration files: Server.xml, Users.xml, Logger.xml, Adaptor.xml, Vhost.xml, and Application.xml. For a description of the XML configuration files, see *Adobe Flash Media Server Configuration and Administration Guide*.

Virtual host administrators can change the values of configuration keys in the Vhost.xml file and Application.xml files for their own virtual hosts. You must be a server administrator to change the values of most configuration keys in the Server.xml and Adaptor.xml files.

**Note:** It is possible to have more than one XML tag with the same name at the same level in the XML tree. In the configuration file, you should distinguish such tags by using a name attribute in the XML tag (for example, if you have more than one virtual host: `<VirtualHost name="www.redpin.com"></VirtualHost>`.) When you call the `setConfig()` method and specify the configuration subkeys, you can indicate which tag you want by specifying the tag name, followed by a colon and the correct name attribute, for example:

`Adaptor:_defaultRoot_/VirtualHost:www.redpin.com.`

**Availability**

Flash Media Server 2.0.

## Parameters

**key** A String indicating the configuration key for which you want to change the value.

A key is specified as a list of subkeys delimited by slashes (/). The first subkey specifies the XML configuration file that contains the desired configuration key. Subsequent subkeys correspond to tags that are relative to that XML configuration file; the hierarchy and names of the subkeys match the tags in the XML file. Here is an example:

```
"/Server/LicenseInfo"
```

**value** A String indicating the value to set for the specified configuration key. If the value specified is a simple string, it is set as the tag data for the specified tag. If the value is valid XML, for example, `<foo>bar</foo>`, the XML is added as a child tag to the specified tag. If the value is an empty string, the specified tag is removed.

**scope** A String indicating the value to set for the tag specified by the **key** parameter. If the value specified is a simple string, it is set as the tag data for the specified tag. If the value is valid XML, for example, `<foo>bar</foo>`, the XML is added as a child tag to the specified tag. If the value is an empty string, the specified tag is removed.

- `/` specifies `Server.xml`. You must have server administrator privileges to access most parts of this file. The only section that does not require server administrator privileges is the `<VirtualHost>` section that contains administrators for the virtual host that the caller belongs to.
- `Users` specifies `Users.xml` for server administrators.
- `Logger` specifies `Logger.xml`.
- `Adaptor:<adaptor_name>` specifies `Adaptor.xml`. The `<adaptor_name>` is the name of the adaptor of interest. You must have server administrator privileges to access this file. If `<adaptor_name>` is not the name of the adaptor the caller is connected to, the call fails.
- `Adaptor:<adaptor_name>/VHost:<vhost_name>` specifies `VHost.xml`. `<vhost_name>` is the name of the virtual host of interest. If `<adaptor_name>` is not the name of the adaptor the caller is connected to, or `<vhost_name>` is not the name of the virtual host that the caller is connected to, the call fails.
- `Adaptor:<adaptor_name>/VHost:<vhost_name>/Users` specifies `Users.xml` for virtual host administrators.
- `Adaptor:<adaptor_name>/VHost:<vhost_name>/App[:<app_name>]` specifies `Application.xml`. If no `<app_name>` is specified, the default `Application.xml` is assumed. Otherwise, the application-specific `Application.xml` file for the specified application is used. If the specified application is not defined, or the application does not have an application-specific `Application.xml` file, the call fails.

**Note:** To determine the adaptor or virtual host to which you're connected, use the `getAdminContext()` method.

**ouser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

## Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status` and a `code` property of `NetConnection.Call.Success`. The value of the configuration key is changed.

If the call fails, the server sends a reply information object with a `level` property of `error`, a `code` property of `NetConnection.Call.BadValue`, and a `description` property that contains a string describing the cause of the failure.

This call fails if the specified configuration key cannot be found or if you do not have permissions to change its value.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
```



```

        <code></code>
        <timestamp></timestamp>
    </result>

```

If the call fails, it returns XML with the following structure:

```

<result>
    <level></level>
    <code></code>
    <description></description>
    <timestamp></timestamp>
</result>

```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## Examples

The following examples show how to set new values for configuration keys:

```

tSocket = new NetConnection();
tSocket.connect("rtmp://localhost/admin", "user", "password");

// set tag data in Server.xml
key = "Server/LicenseInfo";
val = "SFD150-XXXXX-XXXXX-XXXXX";
tSocket.call("setConfig2", new onSetConfig(), key, val, "/");

// set tag data in Adaptor.xml
key = "HostPortList/HostPort";
val = ":1935, 80, 443";
scope = "Adaptor:_defaultRoot_";
tSocket.call("setConfig2", new onSetConfig(), key, val, scope);

// set tag data in Vhost.xml
key = "AppsDir";
val = "c:\\applications";
scope = "Adaptor:_defaultRoot_/VHost:_defaultVHost_";
tSocket.call("setConfig2", new onSetConfig(), key, val, scope);

// set tag data in Application.xml for app "foo"
key = "Process/Scope";
val = "inst";
scope = "Adaptor:_defaultRoot_/VHost:_defaultVHost_/App:foo";
tSocket.call("setConfig2", new onSetConfig(), key, val, scope);

// add child tag to a tag in default Application.xml
key = "Process";
val = "<Scope>inst</Scope>";
scope = "Adaptor:_defaultRoot_/VHost:_defaultVHost_/App";
tSocket.call("setConfig2", new onSetConfig(), key, val, scope);

// delete tag in default Application.xml
key = "Process";
val = "";
scope = "Adaptor:_defaultRoot_/VHost:_defaultVHost_/App";
tSocket.call("setConfig2", new onSetConfig(), key, val, scope);

```

## See also

[getAdminContext\(\)](#), [getConfig2\(\)](#)

# startServer()

**RTMP/E** `startServer([mode:String]) : Object`

**HTTP** `http://www.example.com:1111/admin/startServer?auser=username&apswd=password  
[&mode=string]`

Starts the Flash Media Server service or stops it and restarts it.

You must be a server administrator to perform this operation.

## Availability

Flash Communication Server 1.0.

## Parameters

**mode** A String indicating whether to restart the server.

If the server is not running and you want to start it, omit this parameter. If the server is already running and you want to stop it and then restart it, you must specify `restart` for this parameter. (If the server is running and you omit this parameter, this method does nothing.)

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

## Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status` and a `code` property of `NetConnection.Call.Success`.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Admin.Command.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
  <data></data>
</result>
```

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

## See also

[stopServer\(\)](#)

## startVHost()

**RTMP/E** `startVHost(vhost:String) : Object`

**HTTP** `http://www.example.com:1111/admin/startVHost?auser=username&apswd=password&vhost=name`

Starts a stopped virtual host or enables a new virtual host.

This method lets you enable a new virtual host at run time without restarting the server. The new virtual host directory and configuration files must already be present in the server's conf directory; the `startVHost()` method does not create or copy any directories or configuration files. For information on adding virtual hosts, see *Adobe Flash Media Server Configuration and Administration Guide*.

You must be a server administrator to use the `startVHost()` method.

### Availability

Flash Communication Server 1.0.

### Parameters

**vhost** A String indicating the name of the virtual host you want to start or the new virtual host you want to enable, in the form `[adaptor/]vhost`.

If you are enabling a new virtual host on the adaptor to which you're connected, omit `adaptor/`.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status` and a `code` property of `NetConnection.Call.Success`.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Admin.Command.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
</result>
```

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

### Example

The following example starts a virtual host named `diamond.world.com` on the currently connected adaptor and then starts a virtual host named `diamond.world.com` on the gem adaptor.

```
nc_admin = new NetConnection();
nc_admin.connect("rtmp://localhost:1111/admin", "JLee", "x52z49ab");

/* Starts a virtual host named diamond.world.com */
nc_admin.call("startVHost", new onStartVHost(), "diamond.world.com");

/* Starts a virtual host named diamond.world.com on the adaptor, gem */
nc_admin.call("startVHost", new onStartVHost(), "gem/diamond.world.com");
```

### See also

[restartVHost\(\)](#)

## stopServer()

**RTMP/E** stopServer(mode:String) : Object

**HTTP** `http://www.example.com:1111/admin/stopServer?auser=username&apswd=password  
&mode=string`

Shuts down the Flash Media Server. If you call this method while users are connected, you should take steps to notify users of an imminent server shutdown so that they do not lose their work.

You must be a server administrator to perform this operation.

### Availability

Flash Communication Server 1.0.

### Parameters

**mode** A String indicating how the server is to be shut down. Possible values are `normal` or `abort`. If you use the value `normal`, the server shuts down, allowing running applications to end normally. If you use the value `abort`, the server immediately shuts down and running applications will not be allowed to stop normally. Use the value `abort` only in an emergency or if specifying `normal` does not work.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status` and a `code` property of `NetConnection.Call.Success`.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Admin.Command.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
```

```
</result>
```

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

### See also

[startServer\(\)](#)

## stopVHost()

**RTMP/E** stopVHost([vhost:String]) : Object

**HTTP** http://www.example.com:1111/admin/stopVHost?auser=username&apswd=password  
[&vhost=name]

Stops a running virtual host. (The stopVHost() command cannot be used on the default virtual host.) After the virtual host stops, all applications are unloaded and all users are disconnected. The virtual host does not accept any requests until it has been restarted by means of the startVHost() method or until the server is restarted.

### Availability

Flash Communication Server 1.0.

### Parameters

**vhost** A String indicating the virtual host that you want to stop. To stop the virtual host to which you're connected, omit this parameter.

To stop a different virtual host, specify vhost\_name in the form adaptor\_name/vhost\_name. You must be a server administrator to stop a virtual host other than the one to which you're connected.

**auser** A String indicating the user name of the administrator.

**apswd** A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a level property of status and a code property of NetConnection.Call.Success.

If the call fails, the server sends a reply information object with a level property of error and a code property of NetConnection.Admin.Command.Failed or a more specific value, if available. Some objects might also have a description property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
```

```
<code></code>
<timestamp></timestamp>
</result>
```

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The timestamp response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

### Example

The following example stops the virtual host `tree.oak.com` and then stops the virtual host `tree.oak.com` on the default adaptor.

```
nc_admin = new NetConnection();
nc_admin.connect("rtmp://localhost:1111/admin", "JLee", "x52z49ab");

/* Stop the vhost named tree.oak.com */
nc_admin.call("stopVHost", new onStopVHost(), "tree.oak.com");

/* Stop the vhost named tree.oak.com on the default adaptor */
nc_admin.call("stopVHost", new onStopVHost(), "_defaultRoot_/tree.oak.com");
```

### See also

[restartVHost\(\)](#), [startVHost\(\)](#)

## unloadApp()

**RTMP/E** `unloadApp(appInst:String) : Object`

**HTTP** `http://www.example.com:1111/admin/unloadApp?auser=username&apswd=password&appInst=name`

Shuts down all instances of the specified application or instance of an application. If an application name is specified, all instances of the application are shut down and all the users who are connected to any instance of the application are immediately disconnected. If an instance of an application is specified, only that instance is shut down, and all the users who are connected to that instance are immediately disconnected.

**Note:** Before calling this method, take steps to notify users of the imminent shutdown of an application.

### Availability

Flash Communication Server 1.0.

### Parameters

**appInst** A String indicating the name of the application or instance of the application, in the form `application_name[/instance_name]`.

**auser** A String indicating the user name of the administrator.

`apswd` A String indicating the password of the administrator.

### Returns

**RTMP/E** If the call succeeds, the server sends a reply information object with a `level` property of `status` and a `code` property of `NetConnection.Call.Success`.

If the call fails, the server sends a reply information object with a `level` property of `error` and a `code` property of `NetConnection.Admin.Command.Failed` or a more specific value, if available. Some objects might also have a `description` property that contains a string describing the cause of the failure.

**HTTP** If the call succeeds, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <timestamp></timestamp>
</result>
```

If the call fails, it returns XML with the following structure:

```
<result>
  <level></level>
  <code></code>
  <description></description>
  <timestamp></timestamp>
</result>
```

The XML elements contain the same information as the Object properties returned in an RTMP/E call.

**Note:** The *timestamp* response over HTTP is formatted differently on Windows (9/23/2007 6:16:40 PM) and Linux (Sun 23 Sep 2007 06:16:40 PM IST).

### See also

[reloadApp\(\)](#)