

ADOBE® FLASH® MEDIA SERVER 3.5.3

New Features



Last updated 11/18/2009

© 2009 Adobe Systems Incorporated. All rights reserved.

Adobe® Flash® Media Server 3.5.3 New Features

This guide is licensed for use under the terms of the Creative Commons Attribution Non-Commercial 3.0 License. This License allows users to copy, distribute, and transmit the guide for noncommercial purposes only so long as (1) proper attribution to Adobe is given as the owner of the guide; and (2) any reuse or distribution of the guide contains a notice that use of the guide is governed by these terms. The best way to provide notice is to include the following link. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/>

Adobe, the Adobe logo, ActionScript, and Flash are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. Java is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries. Linux is the registered trademark of Linus Torvalds in the U.S. and other countries. Red Hat is a trademark or registered trademark of Red Hat, Inc. in the United States and other countries. UNIX is a registered trademark of The Open Group in the US and other countries. Windows is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries. Mac, Macintosh, and Mac OS are trademarks of Apple Inc., registered in the United States and other countries. All other trademarks are the property of their respective owners.

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

Contents

New features in Flash Media Server 3.5.3

RAW file format	1
Smart Seek	3
Stream Reconnect	7
Example	11

New features in Flash Media Server 3.5.3

This document provides information about several new features in Flash Media Server 3.5.3: the RAW file format, Smart Seek, and Stream Reconnect. For a complete list of new features, see the ReadMe file. For the complete Flash Media Server documentation, see the [Flash Media Server Help and Support Center](#).

To determine whether the Flash Media Server version is 3.5.3 or higher, check the `info.data.version` information passed in the "NetConnection.Connect.Success" event.

RAW file format

About the RAW file format

Flash Media Server 3.5.3 supports a RAW (Record and Watch) file format that records media into configurable chunks that can be streamed to any version of Flash Player. The RAW file format enables you to serve long-length, multi-bitrate DVR streams without running into performance issues. Use the RAW file format to record and play back all streams that Flash Media Server supports, including H.264 video, data-only, audio-only, and so on.

The RAW file format is a server feature; any version of Flash Player can publish or play a RAW stream. However, multi-bitrate stream support (also called dynamic streaming) requires Flash Player 10 and higher. For more information about dynamic streaming, see [Dynamic streaming](#) in the Flash Media Server *Developer Guide*.

Important: The RAW file format is internal to Flash Media Server. At this time, you cannot edit these files with third-party tools or convert the files to FLV format or MP4 format.

The RAW file format is an FLV file fragmented into the following files:

Filename	Description
index	Contains the list of segment files and their timestamp ranges.
context	Contains all the "context messages" for the stream.
A 16-digit hexadecimal number	There is one file for each segment of the stream. The number of files is the stream index of the first message in the segment. The first segment name is always 0. If the first segment contains 234 messages, the next segment name is EA, and so on.

The files are stored in a folder whose name is the name of the stream. Suppose the stream "foo" is stored in a folder named "foo". The "foo" folder contains the following files: index, context, 0000000000000000, 00000000000001C3, 0000000000000386, and so on.

The RAW file format enables Flash Media Server to handle up to the following scenario:

Parameter	Value
Simultaneous DVR streams	25
Bit rate of each stream	2 Mbit
Codecs	H.264 and AAC
Stream duration	4 hours (x 2 Mbit = 7 GB size)
Number of clients	Depends on origin-edge configuration.

Note: Raw streams with long durations create many files in a single directory. Depending on system resources, the file system may not be able to access the directory fast enough for recording or playback to keep up.

Streaming RAW files

Note: Both ActionScript 2.0 and ActionScript 3.0 support the RAW file format. Flash Media Live Encoder 3.0 does not support the RAW file format.

To record a live stream as a RAW stream, use the prefix `raw:` in the `NetStream.publish()` call or in the `Stream.get()` call.

The following client-side ActionScript uses the RAW file format to publish a live stream:

```
nc:NetConnection = new NetConnection();
nc.connect("rtmp://fms.example.com/live");
// In production code, test for a successful NetConnection here
ns:NetStream = new NetStream(nc);
ns.publish("raw:livestream", "record");
// You can use the "record" or the "append" flag.
```

The following Server-Side ActionScript records a live stream as a RAW stream:

```
s = Stream.get("raw:recordedStream");
s.record();
s.play("livestream", -1, -1);
```

The following client-side ActionScript plays the RAW stream:

```
ns.play("raw:livestream", 0, -1)
```

Validating RAW files and reading error messages

To check if a RAW stream is valid, pass a folder to the FLVCheck tool. For example, if you have a RAW stream named “foo” in the folder “C:\media\foo”, pass the following command to the FLVCheck tool:

```
flvcheck -f C:\media\foo -v -w
```

The following errors are reported by the FLVCheck tool version 2.0 and written to the log files:

Code	Error	Level	Message
-32	Missing/Corrupted Index or Context file	Error	Index or Contexts file missing or corrupted.
-33	Raw file index version doesn't match FMS version	Error	Index File Version %x Not Supported. FMS Requires Version %x.
-34	Segment file missing or corrupted	Error	Failed read in segment file %s. File missing or corrupted.
-35	Truncated message in segment file	Error	Truncated message in segment file %s.
-36	Bad message type	Error	Unrecognized message type in segment file %s.
-37	Message backtag does not match length	Error	Invalid message footer in segment file %s.
-38	Segment data does not match segment bounds	Error	Segment file %s does not match index file.
-119	Contexts file contains no onMetadata message (and there is at least one segment file)	Warning	Missing FLV metadata.
-126	Timestamp decreases from one message to the next	Warning	Found backward timestamp in segment file %s.

FLVCheck version 2.0 adds support for RAW files. FLVCheck version 2.0 is available on the [Flash Media Server software tools](#) page.

Configure the RAW adaptor

- 1 Open the `RootInstall/conf/Server.xml` file in a text editor.
- 2 To configure how the server uses the RAW adaptor, edit the following XML parameters:

```
<Server>
  <Streams>
    <StreamLogLevel>warning</StreamLogLevel>
    <Raw>
      <EnableAggMsgs>true</EnableAggMsgs>
      <MaxAggMsgSize>65536</MaxAggMsgSize>
    </Raw>
  ...
```

Element	Default value	Description
StreamLogLevel	warning	Controls log levels for all stream adaptors (FLV, MP4, and RAW). The default value is <code>warning</code> . Possible values are <code>verbose</code> , <code>warning</code> , and <code>error</code> .
Raw	Container element.	Contains elements that control the RAW adaptor.
EnableAggMsgs	true	Specifies whether the RAW adaptor generates aggregate messages (<code>true</code>) or not (<code>false</code>). The default value is <code>true</code> . Aggregating messages improves server performance.
MaxAggMsgSize	65536	The maximum size of an aggregate message, in bytes. The default value is 65536. You can use any positive integer.

- 3 Save and validate the XML file.
- 4 Restart the server.

Smart Seek

About Smart Seek

Flash Media Server 3.5.3 and Flash Player 10.1 work together to support smart seeking in VOD streams and in live streams that have a buffer. Smart seeking uses back and forward buffers to seek without requesting data from the server. You can step forward and backward a specified number of frames. (Standard seeking flushes buffered data and asks the server to send new data based on the seek time.) Smart seeking reduces server load and improves seeking performance. Use smart seeking to create:

- Client-side DVR functionality. Seek a live stream within the client-side buffer instead of going to the server for delivery of new video.
- Trick modes. Create players that step through frames, fast-forward, fast-rewind, and advance in slow-motion.

Note: Smart seeking is not supported in peer-assisted networking applications or with progressive download

Smart Seek ActionScript API

Note: The Smart Seek ActionScript API is ActionScript 3.0. These APIs are not supported in ActionScript 2.0.

Flash Player maintains a back buffer and a forward buffer. The back buffer is a cache of data that has been displayed. The forward buffer is a cache of data that hasn't been displayed. Smart seeking retrieves data from within these buffers.

To turn on Smart Seek set `NetStream.inBufferSeek` to `true`.

To control the buffers, use the following APIs:

- `NetStream.backBufferLength`
[read-only] The number of seconds of previously displayed data cached for rewinding and playback. This is the `bufferLength` property for the back buffer.
- `NetStream.backBufferTime`
Specifies how much previously displayed data is cached for rewinding and playback, in seconds. The default value is 30 on the desktop and 3 on mobile.
- `NetStream.bufferLength`
[read-only] The number of seconds of data currently in the buffer.
- `NetStream.bufferTime`
Specifies how long to buffer messages before starting to display the stream, in seconds. The default value is 0.1.

To seek and step within the buffers, use the following APIs:

- `NetStream.seek()`
Moves the playhead to the time specified in the call.
- `NetStream.step()`
Steps the playhead forward or back the specified number of frames, relative to the currently displayed frame.

To detect a smart seek, use the following events:

- `NetStatusEvent.info.description` contains the string `"client-inBufferSeek"`.
Dispatched when a call to `NetStream.seek()` is successful.
- `NetStream.Step.Notify`
Dispatched when a call to `NetStream.step()` is successful.

For detailed information about these APIs, see the [ActionScript 3.0 Language Reference](#).

Using the Smart Seek ActionScript API

Smart Seek is supported in Flash Player 10.1 and greater. Before running code, test for Flash Player version. For example, you could expose step back and forward buttons only to clients with Flash Player 10.1 and greater. The following code tests for Flash Player version and returns `true` for 10.1 and greater:

```

public var fp10_1:Boolean;
public function onStart():void{
    debug("Flash Player Version: " + Capabilities.version);
    fp10_1 = isFP10_1();
    debug("fp10.1: "+fp10_1);
}
public function isFP10_1():Boolean {
    var va:Array = Capabilities.version.split(" ")[1].toString().split(",");
    if(int(va[0]) > 10) { return true; }
    if(int(va[0]) < 10) { return false; }
    if(int(va[1]) > 1) { return true; }
    if(int(va[1]) < 1) { return false; }
    return true;
}

```

The following code sets `NetStream.inBufferSeek` to `true` to turn on Smart Seek if the Flash Player version is greater than 10.1:

```

// Call this function when you catch NetConnection.Connect.Success
public function createNetStream():void{

// Write code to create a NetStream object and a Video object...
//...
// Set the forward buffer, in seconds.
ns.bufferTime = 10;
try {
    if(fp10_1) {
// If Flash Player is greater than 10.1, turn on smart seeking
// and set the size of the back buffer, in seconds.
        ns.inBufferSeek = true;
        ns.backBufferTime = 30;
    }
} catch(e:Error) {}

```

When Smart Seek is turned on, calls to `NetStream.seek()` use the buffer. (Standard seeking flushes the buffer and sends a request for data to the server.) The following function can seek forward or backward to a number specified in a text field called `seekText`:

```

public function seekHandler():void {
    if(ns != null) {
        ns.seek(Number(seekText.text));
    }
}

```

To create players that step through frames, fast-forward, fast-rewind, and advance in slow-motion, adjust the number you pass to the `step()` and `seek()` functions.

To create client-side DVR functionality, set the `NetStream.backBufferTime` and `NetStream.bufferTime` properties. These properties specify the amount of data Flash Player stores in the client-side buffer. For example, to allow users to rewind 30 minutes before live, set `backBufferTime` to 1800 (60 seconds x 30 minutes). The cache is stored in memory. Set the buffer properties to lower values if the content is intended for netbooks or mobile devices.

Testing for a Smart Seek

A *Smart Seek* is a call to `NetStream.seek()` or a call to `NetStream.step()` when `NetStream.inBufferSeek` is true.

When a call to `NetStream.seek()` is successful, the `NetStatusEvent.info.description` property contains the string "client-inBufferSeek".

When a call to `NetStream.step()` is successful, the `NetStatusEvent.info.code` property contains the string `"NetStream.Step.Notify"`. If a step has not completed, another call to `step` may return without executing. Get the `"NetStream.Step.Notify"` for the previous call before you call `step` again.

The following code tests for a call to `NetStream.seek()` and a call to `NetStream.step()`:

```
private function netStatusHandler(event:NetStatusEvent):void {
    switch (event.info.code) {
        case "NetStream.Seek.Notify":
            var desc:String = new String(event.info.description);
            if(desc.indexOf("client-inBufferSeek") >= 0)
                trace("A smart seek occurred");
            else
                trace("A standard seek occurred");
            break;
        case "NetStream.Step.Notify":
            trace("Successful NetStream.step() call");
            break;
    }
}
```

Smart Seek requires the following:

- `NetStream.inBufferSeek = true`
The default value of `inBufferSeek` is `false`.
- Flash Media Server 3.5.3
- Flash Player 10.1
- ActionScript 3.0
- The value of the buffers (`backBufferLength` and `bufferLength`) must be large enough to fulfill the seek request.

If any of the previous requirements are not met, Flash Player uses standard seeking but throws no compile-time or runtime errors.

Authorization plug-in events and properties

Use Authorization plug-in events and properties to log information about smart seeking and to block clients from sending Smart Seek commands to the server. The following table summarizes the events and properties for the Smart Seek feature:

Property	Server version	E_CLIENT_SEEK (3.5.3) Notification	E_START_TRANSMIT (3.5.3) Notification and Authorization	E_STOP_TRANSMIT (3.5.3) Notification and Authorization
F_STREAM_SEEK_POSITION	3	Read-only.	None.	None.
F_STREAM_TRANSMIT_POSITION	3.5.3	None.	Read-only.	Read-only.

There are three new events: `E_CLIENT_SEEK`, `E_START_TRANSMIT`, and `E_STOP_TRANSMIT`.

The `E_CLIENT_SEEK` event is a notification event. The Authorization plug-in receives an `E_CLIENT_SEEK` event when a client calls `NetStream.seek()` and `NetStream.inBufferSeek = true` which sends a `seekRaw` command to the server. Use this event to write seek events to a log. You cannot use this event to prevent the client from seeking within the client-side buffer. The `F_STREAM_SEEK_POSITION` property is available for this event. This read-only property is the position to which the client wants to seek.

The `E_START_TRANSMIT` and `E_STOP_TRANSMIT` events are notification and authorization events. Use these events to block the client from asking the server to start or stop sending data.

The Authorization plug-in receives an `E_START_TRANSMIT` event when the client sends a `startTransmit` command. This command asks the server to transmit more data because the buffer is running low. The `F_STREAM_TRANSMIT_POSITION` property is available for this event. This read-only property is the position (in milliseconds) from which the client wants the server to start transmission.

The Authorization plug-in receives an `E_STOP_TRANSMIT` event when the client sends a `stopTransmit` command. This command asks the server to suspend transmission until the client sends a `startTransmit` event because there is enough data in the buffer. The `F_STREAM_TRANSMIT_POSITION` property is available during this event. This read-only property is the position (in milliseconds) of the data at the end of the client buffer when the client sends a `stopTransmit` command.

Server logging

The following events are written to the Flash Media Server `access.log`:

Event	Category	Description
client-seek	stream	The seek position when the client calls <code>NetStream.seek()</code> and <code>NetStream.inBufferSeek = true</code> which sends a <code>seekRaw</code> command to the server. The client sends a <code>seekRaw</code> command only for seeks inside the buffer. If a client seeks outside the buffer, the client sends a "seek" event.
start-transmit	stream	The server received a <code>startTransmit</code> command. This command asks the server to transmit more data because the buffer is running low.
stop-transmit	stream	The server received a <code>stopTransmit</code> command. This command asks the server to suspend transmission until the client sends a <code>startTransmit</code> event because there is enough data in the buffer.

Note: If a user interface lets users step through frames, there could be thousands of calls to `NetStream.step()`. For performance reasons, the server does not write these calls to the log file. These calls do trigger a client-side `"NetStream.Step.Notify" NetStatusEvent`.

Stream Reconnect

About Stream Reconnect

Flash Media Server 3.5.3 and Flash Player 10.1 allow you to build applications that support seamless playback when a connection is dropped or when a client switches from a wired to a wireless network connection.

To provide seamless playback, use the new `NetStream.attach()` method to attach the same `NetStream` object to a reconnected `NetConnection` object. You can also use this feature for load balancing.

Note: If the Flash Player version is less than 10.1 or the Flash Media Server version is less than 3.5.3, the stream closes when the connection drops.

Stream Reconnect ActionScript API

Note: The Stream Reconnect ActionScript API is ActionScript 3.0. These APIs are not supported in ActionScript 2.0.

The following new ActionScript 3.0 APIs enable you to reconnect a stream:

- `NetStream.attach(connection:NetConnection)`
Attaches a stream to a `NetConnection` object.
- `NetStreamPlayTransitions.RESUME`
The `RESUME` mode causes Flash Player 10.1 and greater to request data from a new connection at the same location where it dropped the previous connection. Flash Player aligns the stream across the two connections so no artifacts or jumps are observed in the video playback. Use this mode when you reconnect a stream that was dropped due to server issues or other connection problems.
- `NetStreamPlayTransitions.APPEND_AND_WAIT`
The `APPEND_AND_WAIT` mode tells the server to build the playlist but not to stream it. Use this mode to rebuild a playlist after losing a connection and establishing a new connection.
- `NetConnection.Connect.NetworkChange`
Notifies the client that a network connection has changed. Don't use this event to reconnect a stream, use `NetConnection.Connect.Closed`.

For detailed information about these APIs, see the [ActionScript 3.0 Language Reference](#).

Using the Stream Reconnect ActionScript API

When a `NetConnection` closes due to a network change, the stream keeps playing using the existing buffer. Meanwhile, client-side ActionScript code reconnects to the server and resumes playing the stream.

Note: If the Flash Player version is less than 10.1 or the Flash Media Server version is less than 3.5.3, the stream closes as soon as the network connection is lost.

Reconnecting a single stream

The following workflow reconnects a single stream:

- 1 Call `NetConnection.connect()` to connect to server A.
- 2 Create a `NetStream`. Set `NetStream.bufferTime` to at least a few seconds so there is enough data to play while the connection is down.
- 3 Call `NetStream.play2()` and use `NetStreamPlayTransitions.RESET` to play a stream called "myStream".
- 4 Monitor the `NetConnection` for the "`NetConnection.Connect.Closed`" event. Reconnect to server A if the connection drops.
- 5 Call the `NetStream.attach(connection:NetConnection)` method to attach the `NetStream` to the new connection.
- 6 Call `NetStream.play2()` and use `NetStreamPlayTransitions.RESUME` to play the stream "myStream".
Flash Player tells the server where to resume playing the stream.
- 7 Do not call `NetConnection.close()` or `NetStream.close()` if you intend to reconnect it. Calling `close` closes and cleans up the connection or the stream, so it will stop playback right away and future reconnects will not be possible.
After a `attach` is done to the new `NetConnection`, a `close` on the older `NetConnection` can be called.

Reconnecting a playlist

The following workflow reconnects a playlist:

- 1 Call `NetConnection.connect()` to connect to server A.
- 2 Create a `NetStream`. Set `NetStream.bufferTime` to at least a few seconds so there is enough data to play while the connection is down.
- 3 Call `NetStream.play2()` and use `NetStreamPlayTransitions.RESET` to play a stream called “myStream1”.
- 4 Call `NetStream.play2()` and use `NetStreamPlayTransitions.APPEND` to play a second stream called “myStream2”.
- 5 Monitor the `NetConnection` for the “`NetConnection.Connect.Closed`” event. Reconnect to server A if the connection drops.
- 6 Call the `NetStream.attach(connection:NetConnection)` method to attach the `NetStream` to the new connection.
- 7 Call `NetStream.play2()` and use `NetStreamPlayTransitions.APPEND_AND_WAIT` to add “myStream1” to the playlist.
- 8 Call `NetStream.play2()` and use `NetStreamPlayTransitions.RESUME` to add “myStream2” to the playlist and resume the stream.

Flash Player tells the server where to resume playing the stream.

Balancing a server load

The following workflow uses the smart reconnect API to balance a load:

- 1 Call `NetStream.attach(connection:NetConnection)` to attach a stream to a `NetConnection` object on another server.
- 2 After the stream is successfully attached to the new connection, call `NetConnection.close()` on the prior connection to prevent data leaks.
- 3 Call `NetStream.play2()` and use `NetStreamPlayOptions.transition` to `RESUME`. Set the rest of the `NetStreamPlayOptions` properties to the same values you used when you originally called `NetStream.play()` or `NetStream.play2()` to start the stream.

Monitoring a network interface change

To monitor changes to the network interface, monitor the `NetConnection` for the “`NetConnection.Connect.NetworkChange`” event. This event detects a network change (for example, connecting to a wireless network, disconnecting from a wireless network, disconnecting a network cable, and so on).

Note: Do not use this event to monitor a dropped `NetConnection` or to write logic for to reconnect a stream. Use “`NetConnection.Connect.Closed`”.

Monitoring a connection on a mobile device

Some mobile devices cannot receive a “`NetConnection.Connect.Closed`” message. In this case, you can monitor the `NetStream.bufferLength` and `NetStreamInfo.byteCount` properties in a timer to discover network issues. When `NetStream.bufferLength` is less than `NetStream.bufferTime` and `NetStreamInfo.byteCount` is not increasing, there are probably network issues.

```

netStreamMonitorTimer.start();
netStreamMonitorTimer.addEventListener(TimerEvent.TIMER, timerHandler);
lastByteCount = 0;
private function timerHandler(e:TimerEvent):void{
    if(netstream.bufferLength < netstream.bufferTime && netstream.info.byteCount ==
lastByteCount) {
        // Network has issues.. reconnect to a new NetConnection
        netconnection2 = new NetConnection(); //on NetConnection.Connect.Success:
        netStream.attach(netconnection2);
    }
    lastByteCount = netstream.info.byteCount;
}

```

Authorization plug-in events and properties

Use the `E_PLAY` event of the Authorization plug-in to control streaming that occurs after a reconnection. The following table summarizes the events and properties for the Stream Reconnect feature:

Property	Server version	E_PLAY Notification and Authorization
F_STREAM_OFFSET	3.5.3	Read-only.
F_STREAM_TRANSITION	3.5	Read and write.

There is one new property: `F_STREAM_OFFSET`. The `F_STREAM_OFFSET` property indicates where to resume streaming after a reconnection, in seconds.

The `F_STREAM_TRANSITION` property indicates the transition mode sent by the client in the `NetStream.play2()` call. The values for Stream Reconnect are “resume” and “appendAndWait”.

Server logging

The following events are written to the Flash Media Server access log for the Stream Reconnect feature:

Event	Category	Description
connect	session	A client has connected to a Flash Media Server application. This event is logged when you re-establish a connection after it drops.
play	stream	A stream has resumed playing.
stop	stream	A stream has stopped playing.

The following fields are new for the Stream Reconnect feature:

Field	Description
x-trans-mode	The transition mode sent by the client in the <code>NetStream.play2()</code> call. For Stream Reconnect, the transition modes are “resume” and “appendAndWait”.
x-offset	The offset value indicates where to resume streaming after you attach a <code>NetStream</code> .

These fields are disabled by default. You can optionally display these fields in the `authEvent.log` and in the `access.log`.

Example

The following is a hosted example of the Stream Reconnect and Smart Seek features:

aspeexamples.adobe.com/flash/streamreconnect.html. Right-click on the example and choose View Source to view and download the source files.