# Using
# ADOBE® HTTP DYNAMIC STREAMING

Using Adobe® HTTP Dynamic Streaming



Last updated 6/30/2010

# Contents

# Chapter 1: Getting Started

Adobe® HTTP Dynamic Streaming is a solution that allows you to stream live and on-demand content over HTTP to Adobe® Flash® Player 10.1. When content streams over HTTP, clients can seek quickly to any location. HTTP Dynamic Streaming supports adaptive streaming, DVR functionality, and Adobe® Flash® Access™ protection.

## Overview

### About HTTP streaming

Delivering content over HTTP is called "progressive download". The content must transfer from the server to the client in a progression from the beginning to the end of a file. A client cannot seek to a forward location until that location and all the data before it has downloaded.

Delivering content over RTMP is called "streaming". The media server (such as Flash Media Server) creates a socket connection to the client over which the content is sent in a continuous stream. The client can seek to any point in the content instantly, regardless of how much data has been transferred.

HTTP Dynamic Streaming combines these approaches to introduce *HTTP streaming* to the Flash Platform. HTTP Dynamic Streaming packages media files into fragments that Flash Player clients can access instantly without downloading the entire file. Adobe HTTP Dynamic Streaming contains several components that work together to package media and stream it over HTTP to Flash Player.

### Components

The Adobe HTTP Dynamic Streaming solution contains the following components:

**File Packager** A command-line tool that translates on-demand media files into fragments and writes the fragments to F4F files. The File Packager is an offline tool. You can use the File Packager to encrypt files for use with Flash Access. For more information, see "Packaging on-demand media" on page 20.

The File Packager is available from adobe.com and is installed with Adobe® Flash® Media Server to the *rootinstall*/tools/f4fpackager folder.

**Live Packager for HTTP Dynamic Streaming** The Live Packager for HTTP Dynamic Streaming is part of Adobe Flash Media Server 3.8 and later. The server ingests a live stream over RTMP and translates it into F4F files in real-time. The built-in Apache HTTP Server uses the HTTP Origin Module to deliver the live content over HTTP. For more information, see "Packaging live media" on page 29.

For information about downloading Flash Media Server 3.8, contact Adobe.

**HTTP Origin Module** An Apache HTTP Server module that serves the on-demand files created by the File Packager and the live streams created by Flash Media Server. For more information, see "Installing and configuring the HTTP Origin Module" on page 13.

The HTTP Origin Module is available from adobe.com and is installed with Adobe® Flash® Media Server 3.8 and later.

**OSMF Player** Built on the Open Source Media Framework (OSMF), this video player runs in Flash Player 10.1. For more information, see "Using the OSMF Player" on page 5.

Download the OSMF Player from osmf.org.

**F4F File Format Specification** The F4F file format describes how to divide media content into segments and fragments. Each fragment has its own bootstrap information that provides cache management and fast seeking. For more information, see F4F File Format Specification.

**F4M File Format Specification** The Flash Media Manifest file format contains information about a package of files that the HTTP Origin Module can serve. Manifest information includes codecs, resolutions, and the availability of files encoded at multiple bit rates. Manifest information also includes DRM data. For more information, see F4M File Format Specification.

**Flash Access** Flash Access delivers protected media to Flash Player. To use HTTP Dynamic Streaming with Flash Access, use the File Packager and Flash Media Server to both package and encrypt content. For more information, see Protecting content with Flash Access.

## System requirements

The following are the system requirements for HTTP Dynamic Streaming:

| Component | Requirement |
|---|---|
| Flash Media Interactive Server 3.8 | This version of Flash Media Server supports live and on-demand HTTP Dynamic Streaming. This version is a 32-bit binary that runs on the following operating systems:<br>• Microsoft® Windows Server® 2008 32-bit<br>• Red Hat® Linux® 5.1 32-bit<br>**Note:***The Windows version of the server includes the Microsoft® Visual C++ 2005 Redistributable Package and the Microsoft Visual C++ 2008 Redistributable Package.* |
| File Packager | The File Packager is a 32-bit binary that runs on the following operating systems:<br>• Microsoft® Windows Server® 2008 32-bit<br>• Red Hat® Linux® 5.1 32-bit |
| HTTP Origin Module | The HTTP Origin Module is a 32-bit binary extension of Apache HTTP Server version 2.2, 32-bit. The HTTP Origin Module and Apache HTTP Server run on the following operating systems:<br>• Microsoft® Windows Server® 2008 32-bit<br>• Red Hat® Linux® 5.1 32-bit<br>The stand-alone HTTP Origin Module supports on-demand HTTP Dynamic Streaming only. It does not support live streaming.<br>**Important:***To run the stand-alone HTTP Origin Module on Windows, download and install Microsoft Visual C++ 2005 Redistributable Package and Microsoft Visual C++ 2008 Redistributable Package.* |
| OSMF Player | OSMF 1.0 |

| Component | Requirement |
|---|---|
| Runtimes | Flash Player 10.1, AIR 2 |
| Flash Access | 2.0 |
| File formats and codecs | The File Packager and Live Packager convert F4V/MP4 and FLV files to F4F files.<br><br>HTTP Dynamic Streaming supports the H.264, VP6, MP3, and AAC codecs that Flash Player supports. If a file plays in Flash Player, HTTP Dynamic Streaming (with or without Flash Access) can package it and serve it.<br><br>**Important:** *HTTP Dynamic Streaming 1.0 does not support the MP3 file format or Speex audio.* |

## Use cases

HTTP Dynamic Streaming supports the following use cases:

- On-demand content
- On-demand multi-bitrate content (also called “adaptive streaming” and “dynamic streaming”)
- Live content
- Live multi-bitrate content
- Live content with DVR functionality
- Live multi-bitrate content with DVR functionality

HTTP Dynamic Streaming also supports each of the previous use cases with Flash Access content protection.

## On-demand workflow

The following is the workflow for all on-demand streaming use cases:

1 Install and configure the HTTP Origin Module. Do one of the following:

- Install Flash Media Interactive Server 3.8.
- Install the HTTP Origin Module to any installation of Apache HTTP Server 2.2.

See “Installing and configuring the HTTP Origin Module” on page 13.

2 Run the File Packager to process media files. Do one of the following:

- To package a single media file, see “Package content on Windows” on page 21 or “Package content on Linux” on page 21.
- To package multi-bitrate content, use the File Packager to create a manifest file that contains information about each encoded media file. See “Packaging multi-bitrate content” on page 22.
- To encrypt content for Flash Access, specify encryption options. See “Package and encrypt content” on page 23.

3 Copy the files output from the File Packager to a location on the Apache HTTP server under the DocumentRoot folder.

The DocumentRoot folder is the folder from which Apache serves files.

4 Create a crossdomain.xml file and copy it to the Apache DocumentRoot folder.

A crossdomain.xml file allows Flash Player clients hosted on other domains to access data from this domain. For more information, see Website controls (policy files) in the ActionScript 3.0 Developer’s Guide.

5 Use the OSMF Player to play the media in Flash Player 10.1.

See "Using the OSMF Player" on page 5

## Live workflow

The following is the workflow for all live streaming use cases:

1 Install Flash Media Interactive Server 3.8.

2 To configure the Live Packager to encrypt content for use with Flash Access, edit the Event.xml file. See "Event.xml file" on page 34.

3 Create a crossdomain.xml file and copy it to the Flash Media Interactive Server *rootinstall*\webroot folder.

A crossdomain.xml file allows Flash Player clients hosted on other domains to access data from this domain. For more information, see Website controls (policy files) in the ActionScript 3.0 Developer's Guide.

4 Publish a stream to Flash Media Interactive Server.

See "Publish a stream to the Live Packager and associate the stream with a live event" on page 30.

5 Use the OSMF Player to play the media in Flash Player 10.1.

See "Using the OSMF Player" on page 5

## Flash Media Server default HTTP Dynamic Streaming configurations

Flash Media Server 3.8 is configured by default for live and on-demand HTTP Dynamic Streaming.

| Asset type | Description |
|---|---|
| Flash Media Server application | *rootinstall*/applications/livepkgr |
| Live event name for a single stream | *rootinstall*/applications/livepkgr/events/_definst_/<stream-name> |
| Live event name for multi-bitrate streams | *rootinstall*/applications/livepkgr/events/_definst_/liveevent |
| Live stream name for a single stream | livestream |
| Live stream name for multi-bitrate streams | livestream%i?adbe-live-event=liveevent&adbe-http-streaming-ver=1.0 |
| File location for on-demand streaming | *rootinstall*/webroot/vod |

The Apache HTTP Server is configured to expect the following URLs from OSMF Player:

| Request type | Syntax |
|---|---|
| Live | http://<server>/live/events/livepkgr/events/_definst_/liveevent.f4m |
| On-demand | http://<server>/vod/<vodasset>.f4m |
| Live bootstrap file | http://<server>/live/streams/livepkgr/streams/_definst_/<streamname>/<streamname>.bootstrap |
| Live fragment | http://<server>/live/streams/livepkgr/streams/_definst_/<streamname>/<streamname>SegN-FragM |

## Using the OSMF Player

The OSMF Player is a video player built on the Open Source Media Framework (OSMF). The OSMF Player runs in Flash Player 10.1.

The OSMF Player uses the ActionScript 3.0 `NetStream.appendBytes()` API to deliver bytes to Flash Player. To use this API to create an HTTP Dynamic Streaming player, use the OSMF Player as a reference implementation. However, OSMF is a robust framework designed to deliver high-quality video. Adobe strongly recommends using OSMF to build HTTP Dynamic Streaming players.

**Access the OSMF Player at the following locations:**

- OSMF Player for HTTP Dynamic Streaming
- OSMF Player source
- Hosted OSMF Player

**To host OSMF Player on a web server:**

1 Download the OSMF Player.

2 Copy the following files to a web server:

- OSMFPlayer.html
- OSMFPlayer.swf
- The \assets folder and the contained files.
- The \images folder and the contained files.
- The \scripts folder and the contained files.

**To load media into the OSMF Player from a URL:**

❖ Click the Eject button, enter the URL, and click Enter.

The configuration of the HTTP Origin Module determines the format of the URL. For information about how to form a request URL, see "Configure on-demand streaming" on page 15 and "Configure live streaming" on page 16.

**To test dynamic streaming:**

1 Click "Q" to toggle between automatic and manual bitrate switching.

When the player is in manual mode, a + and - button display.

2 In manual mode, click + and - to increase and decrease the bitrate of the stream.

### More Help topics

"Tutorial: On-demand HTTP Dynamic Streaming" on page 6

"Tutorial: Live HTTP Dynamic Streaming" on page 9

## Protecting content with Flash Access

To use deliver content with HTTP Dynamic Streaming and protect it with Flash Access, use the following tools:

- Flash Access Server for Protected Streaming

  This server is a Flash Access license server implementation optimized for use with HTTP Dynamic Streaming. See *Adobe Flash Access Protecting Content*.

***Note:*** *The Flash Access SDK and the Flash Access license server reference implementation can also issue licenses for HTTP Dynamic Streaming.*

- HTTP Dynamic Streaming File Packager

  Use this tool to package content offline for on-demand streaming.

- Live Packager for HTTP Dynamic Streaming (part of Flash Media Server)

  Use this tool to packager content in real-time for live streaming.

The certificates that you use with the File Packager and the Live Packager must match the license server.

***Important:*** *Do not use the Flash Access packaging tools to encrypt content. The HTTP Dynamic Streaming packagers cannot fragment content that has already been encrypted. Use the HTTP Dynamic Streaming packagers to both encrypt and fragment content.*

**Workflow for using Flash Access with live HTTP Dynamic Streaming**

1. A camera captures a live event and sends the output to an encoder such as Flash Media Live Encoder.
2. The encoder sends a stream to the Live Packager for HTTP Dynamic Streaming which is part of Flash Media Server.

   For information about configuring Flash Media Live Encoder to send a stream to the Live Packager, see "Tutorial: Live HTTP Dynamic Streaming" on page 9.
3. Configure the Live Packager to fragment and encrypt the content. The Live Packager processes the content in real time. To configure the Live Packager to encrypt content, edit the Event.xml file. See "Event.xml file" on page 34.
4. The Live Packager outputs the fragmented and protected files needed for HTTP Dynamic Streaming. Each fragment is persistently protected in both the CDN cache and the browser cache.
5. The OSMF Player does the following:
   a. Requests the file fragments and plays them back seamlessly.
   b. Obtains a content license from Flash Access Server for Protected Streaming. The content license contains the key required to play the content to legitimate clients.
   c. Decrypts the content in real-time.

**Workflow for using Flash Access with on-demand HTTP Dynamic Streaming**

1. Use the File Packager to package and encrypt content. See "File Packager reference" on page 24.
2. Copy the packaged files to an Apache HTTP Server that uses the HTTP Dynamic Streaming Origin Module.
3. The OSMF Player does the following:
   a. Requests the file fragments and plays them back seamlessly.
   b. Obtains a content license from Flash Access Server for Protected Streaming. The content license contains the key required to play the content to legitimate clients.
   c. Decrypts the content in real-time.

# Quick start

## Tutorial: On-demand HTTP Dynamic Streaming

**Requirements**

To complete this tutorial, download the following software:

- OSMF Player for HTTP Dynamic Streaming
- Flash Player 10.1

On-demand HTTP Dynamic Streaming requires Apache 2.2. Apache 2.2 is included with Flash Media Server 3.5 and later. Download one of the following:

- Apache 2.2
- Flash Media Server 3.5
- Flash Media Server 3.8

On-demand HTTP Dynamic Streaming requires software tools included with Flash Media Server 3.8. If you downloaded Apache 2.2 or Flash Media Server 3.5, download the following:

- File Packager and HTTP Origin Module

Follow these steps to create your first end-to-end, on-demand, multi-bitrate HTTP Dynamic Streaming application:

1 Do one of the following to install Apache:

- Install Flash Media Server 3.8 or later.

  This version of the server includes Apache and the HTTP Origin Module. On Windows, it includes the Microsoft Visual C++ 2005 Redistributable Package and the Microsoft Visual C++ 2008 Redistributable Package. On Linux, it includes the Linux libraries.

- Install Flash Media Server 3.5.

  This version of the server includes Apache. This version does not include the HTTP Dynamic Streaming Origin Module, the Visual C++ runtime packages, or the Linux libraries.

- Install Apache HTTP Server version 2.2.

  This version of Apache doesn't include the HTTP Origin Module, the Visual C++ runtime packages, or the Linux libraries.

2 If you're using Apache with Flash Media Server, "Configure Apache and Flash Media Server to run on the same computer" on page 14.

3 If you're not using Flash Media Server 3.8, do the following:

a On Windows, download and install Microsoft Visual C++ 2005 Redistributable Package and Microsoft Visual C++ 2008 Redistributable Package.

On Linux, download and install the following libraries: OpenSSL, Expat, and Netscape Portable Runtime (NSPR).

b Install the HTTP Origin Module.

On Windows, copy the following files to the Apache \modules folder: mod_f4fhttp.so, adbe_F4V.dll, and libexpat.dll.

On Linux, copy the following files to the Apache \modules folder: mod_f4fhttp.so, and libF4V.so file.

c Create a "vod" folder in the Apache DocumentRoot folder. The vod folder will hold the packaged media files.

On Windows, the default location is C:\Program Files\Apache Software Foundation\Apache2.2\htdocs\vod.

On Linux, the default location is /usr/local/apache/htdocs/vod.

**d** To configure the HTTP Origin Module, do the following:

**e** Open the Apache \conf\httpd.conf file in a text editor.

**f** Locate the `LoadModule` section and add the following line:

```
LoadModule f4fhttp_module modules/mod_f4fhttp.so
```

***Note:*** *If there isn't a `LoadModule` section, add the line anywhere in the file.*

**g** Locate the `<Location>` section and add the following:

```
<Location /vod>
    HttpStreamingEnabled true
    HttpStreamingContentPath "C:\Program Files\Apache Software
Foundation\Apache2.2\htdocs\vod"
</Location>
```

***Note:*** *Set the `HttpStreamingContentPath` directive to the full path of the "vod" folder you created in step 2c.*

**h** Save and close the file.

**i** Restart Apache HTTP Server.

**4** To package files, complete the task "Package multi-bitrate files" on page 22 and return to this tutorial.

**5** Copy the packaged files to the "vod" folder you created in step 3c.

On Flash Media Server, the "vod" folder is in the following locations:

- On Windows, C:\Program Files\Adobe\Flash Media Server 3.8\webroot\vod.
- On Linux, /opt/adobe/fms/webroot/vod.

**6** To play the media in OSMF Player, do the following:

**a** Copy the OSMF Player files (including the \scripts, \images, and \assets folders and their files) to the DocumentRoot folder. This folder is the parent of the "vod" folder.

***Note:*** *If the DocumentRoot folder already contains an "images" folder, copy the files in the OSMF Player \images folder to the DocumentRoot\images folder.*

**b** Open a web browser and enter the address of the OSMF Player.

This tutorial uses **http://localhost/OSMFPlayer.html**.

***Note:*** *OSMF Player requires Flash Player 10.1.*

**c** Click the Eject button and enter **http://localhost/vod/sample1_1500kbps.f4m**.

**d** Click Play.

**e** To test multi-bitrate streaming, click the Q button to turn on manual switching. Click Q- and Q+ to shift down and up.

## Understanding the on-demand tutorial

To play a video in OSMF Player, request an F4M file, also called a *manifest file*. The request syntax for on-demand streaming is:

```
http://<server-name>/<location-tag-alias>/<manifest-file-name>.f4m
```

***Note:*** *The syntax is the same for all on-demand use cases.*

The following table explains the variables in the request syntax:

| Variable | Description |
|---|---|
| <server-name> | The domain name or IP address of the computer hosting Flash Media Server. In this tutorial, OSMF Player and Flash Media Server are on the same computer, so the value is `localhost`. |
| <location-tag-alias> | This value is configured in a Location directive in the Apache /conf/httpd.conf file. The default value for on-demand http streaming in the Apache HTTP Server that installs with Flash Media Server is `vod`. |
| <manifest-file-name> | The name of the manifest (.f4m) file created by the File Packager. |

### More Help topics

"Configure on-demand streaming" on page 15

"Requesting on-demand files" on page 15

## Tutorial: Live HTTP Dynamic Streaming

**Requirements**

To complete this tutorial, download the following software:

- Flash Media Server 3.8 or later
- Flash Media Live Encoder
- OSMF Player for HTTP Dynamic Streaming
- Flash Player 10.1

**Follow these steps to create your first end-to-end, live HTTP Dynamic Streaming application:**

1 Install Flash Media Interactive Server 3.8 and choose to install Apache HTTP Server 2.2.

2 Install Flash Media Live Encoder and configure it to use absolute time.

**a** Close Flash Media Live Encoder.

**b** Open the Flash Media Live Encoder *rootinstall*\Conf\config.xml file in a text editor.

**c** Set the tag //flashmedialiveencoder_config/mbrconfig/streamsynchronization/enable to true:

```
<flashmedialiveencoder_config>
    <mbrconfig>
        <streamsynchronization>
            <!-- "true" to enable this feature, "false" to disable.-->
            <enable>true</enable>
```

**d** Save the file.

3 To publish a live stream to Flash Media Server, start Flash Media Live Encoder and do the following:

**a** In the Encoding Options panel, from the Preset pop-up menu, choose Multi Bitrate - 3 Streams (1500 Kbps) - H.264.

**b** Click the wrench next to Format to open Advanced Encoder Settings. For Keyframe frequency, select 4 seconds.

***Note:*** *This value matches the <FragmentDuration> value in the applications/livepkgr/events/_definst_/liveevent/Event.xml file. The <FragmentDuration> value is in milliseconds.*

**c** For Bit Rate, choose 100, 200, and 350.

***Note:*** *These values match the <media bitrate> values in the applications/livepkgr/events/_definst_/liveevent/Manifest.xml file.*

**d** In the FMS URL text box, enter **rtmp://localhost/livepkgr**.

***Note:*** *This application is installed with Flash Media Server and contains a main.asc file and configuration files for live HTTP Dynamic Streaming.*

**e** In the Stream text box, enter **livestream%i?adbe-live-event=liveevent&adbe-http-streaming-ver=1.0**.

***Note:*** *The applications/livepkgr/main.asc file expects this value.*

**f** Deselect Save to File.

**g** Click Start.

**4** To play the media in the OSMF Player, do the following:

**a** Copy the OSMF Player files to the *rootinstall*\webroot folder (including the \scripts, \images, and \assets folders and their files). The webroot folder already contains an \images folder so copy the files from the OSMF Player \images folder to the existing \images folder.

**b** Open a web browser and enter the address of the OSMF Player: **http://localhost/OSMFPlayer.html**.

***Note:*** *OSMF Player requires Flash Player 10.1.*

**c** Click the Eject button, enter **http://localhost/live/events/livepkgr/events/_definst_/liveevent.f4m**, and press Enter on your keyboard.

## Tutorial: Live HTTP Dynamic Streaming with DVR

To add DVR capabilities to a live HTTP stream, complete the "Tutorial: Live HTTP Dynamic Streaming" on page 9 with two additional steps:

**1** Complete Step 1 to install Flash Media Server.

**2** To configure the server to expect a DVR stream, do the following:

**a** Open the file *fmsrootinstall*/applications/livepkgr/events/_definst_/liveevent/Manifest.xml in a text editor.

**b** Add the `<dvrInfo>` tag. The file should look like this:

```
<manifest xmlns="http://ns.adobe.com/f4m/1.0">
 <dvrInfo beginOffset="0" endOffset="0"></dvrInfo>
 <media streamId="livestream1" bitrate="100" />
 <media streamId="livestream2" bitrate="200" />
 <media streamId="livestream3" bitrate="350" />
</manifest>
```

**c** Save and close the file.

**3** Complete Step 2 to configure Flash Media Live Encoder to use absolute time.

**4** Complete Step 3 to stream video from Flash Media Live Encoder. To add DVR capabilites, select DVR Auto Record.

**5** Complete Step 4 to play the video in OSMF Player.

## Understanding the live tutorial

### Understanding the request URL

To play a video in OSMF Player, request an F4M file, also called a *manifest file*.

***Important:*** *For live streaming, the HTTP Origin Module generates the F4M file in real-time. The F4M file is not written to disk, you cannot see it on the server.*

The request syntax for multi-bitrate streaming is:

```
http://<server-name>/<location-tag-alias>/events/<fms-app-name>/events/<fms-app-inst-name>/<live-event-name>.f4m
```

The following table explains the variables in the request syntax:

| Variable | Description |
|---|---|
| <server-name> | The domain name or IP address of the computer hosting Flash Media Server. In this tutorial, OSMF Player and Flash Media Server are on the same computer, so the value is `localhost`. |
| <location-tag-alias> | This value is configured in a Location directive in the Apache /conf/httpd.conf file. The default value for live http streaming is `live`. |
| <fms-app-name> | The name of the Flash Media Server application to which live streams are published. Live streams published to this application are processed by the Live Packager. The name of the application is `livepkgr`. |
| <fms-app-inst-name> | The name of the Flash Media Server application instance. The default application instance is `_definst_`. |
| <live-event-name> | The name of the live event within the Flash Media Server application. The default live event is `liveevent`. When you publish a single stream (not multiple streams encoded at different bit rates), the default live event name is the same as the stream name. |
| <live-stream-name> | The name of the stream published to the Flash Media Server application. |

This tutorial uses multi-bitrate streaming. However, you may choose to publish a single stream instead. The request syntax for a single stream is:

```
http://<server-name>/<location-tag-alias>/events/<fms-app-name>/events/<fms-app-inst-name>/<live-stream-name>.f4m
```

To publish a single stream, in Flash Media Live Encoder, choose a single stream Preset. In the Stream text box, enter livestream%i.

### More Help topics

"Configure live streaming" on page 16

"Requesting live streams" on page 17

## Files created by the Live Packager

To play a stream in OSMF Player, request a .f4m file. OSMF Player contains logic that requests additional files when a user seeks, pauses, and plays media. The Live Packager creates these files when the server ingests a live stream. You don't need to do anything with these files, but you may want to know where they're located on the server.

In the *rootinstall*\applications\livepkgr\events_definst_\liveevent folder, the Live Packager creates a .stream file for each live stream published to the livepkgr application. The name of the .stream file is encoded, for example, MzgyODc4OTAyMg=.stream. Open the file in a text editor to see the content.

In the *rootinstall*\applications\livepkgr\streams_definst_ folder, the Live Packager creates a folder with the name of each stream, livestream1, livestream2, and livestream3. The Live Packager creates the following files in each folder:

- livestream#.bootstrap

- livestream#.control
- livestream#.meta
- livestream#Seg#.f4f
- livestream#Seg#.f4x

### More Help topics

"Live Packager reference" on page 33

# Chapter 2: Installing and configuring the HTTP Origin Module

The HTTP Origin Module is an extension to Apache HTTP Server that serves content packaged for HTTP Dynamic Streaming. The HTTP Origin Module is available from adobe.com/products/httpdynamicstreamingand is installed with Flash Media Interactive Server 3.8.

## About the HTTP Origin Module

### HTTP Origin Module requirements

The HTTP Origin Module is an extension to Apache HTTP Server version 2.2. The HTTP Origin Module processes F4F, F4M, F4X, .bootstrap and .drmmeta files. For more information about these file types, see "Input file formats and output file formats" on page 20.

Flash Media Interactive Server 3.8 includes Apache HTTP Server and the HTTP Origin Module and is configured for HTTP Dynamic Streaming.

The HTTP Origin Module reference implementation available on adobe.com supports on-demand streaming only. You can install this HTTP Origin Module on any installation of Apache HTTP Server 2.2.

### Data flow through the HTTP Origin Module

To play a file in the OSMF Player, you provide only the URL to an F4M file. The OSMF Player automatically handles the transactions in steps 4-8.

1. The client sends an HTTP request for media to Apache, for example:

   ```
   http://www.example.com/media/http_dynamic_Streaming.f4m
   ```

2. Apache passes the request to the HTTP Origin Module.
3. The HTTP Origin Module returns the F4M file to the client.
4. The client receives the F4M file and uses the bootstrap information to translate time code into a segment#/fragment# pair.
5. The client sends a second HTTP request to Apache and asks for a specific fragment from the F4F file, for example:

   ```
   http://www.example.com/media/http_dynamic_StreamingSeg1-Frag1
   ```

6. Apache receives the request and passes the request to the HTTP Origin Module.
7. The HTTP Origin Module uses the index file (F4X) to translate the request into a byte offset in the F4F file. It sends the contents of this file that correspond to Segment1 and Fragment. This partial content is an F4F fragment, for example:

   ```
   http://www.example.com/media/http_dynamic_StreamingSeg1.f4f
   ```

8. The client receives the fragment and starts playback based on the bootstrap information provided with the fragment and the information from the F4M file.

## Requesting files from the HTTP Origin Module

To request a file from the HTTP Origin Module, see the following:

- "Requesting live streams" on page 17
- "Requesting on-demand files" on page 15

## Configure Apache and Flash Media Server to run on the same computer

The Flash Media Server installer includes Apache HTTP Server. By default, Flash Media Server receives all traffic on port 80 and proxies it to Apache on port 8134. Proxying the traffic can cause performance issues with HTTP Dynamic Streaming because of the high number of HTTP requests at both edge and origin network locations.

To fix this issue, configure Flash Media Server not to use port 80 and configure Apache to use port 80.

**Configure Flash Media Server not to use port 80:**

1. Open the *rootinstall*/conf/fms.ini file in a text editor.
2. Remove port 80 from the ADAPTOR.HOSTPORT parameter, as follows:

   ```
   ADAPTOR.HOSTPORT = :1935
   ```

3. Remove the value from the HTTPPROXY.HOST parameter as follows:

   ```
   HTTPPROXY.HOST =
   ```

4. Save and close the fms.ini file.
5. Restart the server. See Starting and stopping the server.

**Configure Apache to use port 80:**

1. Open the *rootinstall*/Apache2.2/conf/httpd.conf file in a text editor.
2. Locate the line `Listen 8134` and change it to the following:

   ```
   Listen 80
   ```

3. Save and close the httpd.conf file.
4. Restart the server. See Starting and stopping the server.

***Note:*** *Alternately, you can configure Apache to use a port other than port 80. To use a port other than 80, request URLs must contain the port number. By default, Apache on Flash Media Server is configured to use port 8134. This URL requests an on-demand HTTP Dynamic Streaming manifest file: http://www.example.com:8134/vod/mymedia.f4m.*

# Installing and configuring the HTTP Origin Module

## Prerequisites

1. Install Apache HTTP Server version 2.2 or later.
2. On Windows, download and install Microsoft Visual C++ 2005 Redistributable Package and Microsoft Visual C++ 2008 Redistributable Package.

   On Linux, download and install the following libraries: OpenSSL, Expat, and Netscape Portable Runtime (NSPR).

## Install the HTTP Origin Module

❖ Copy the following files to the Apache \modules folder:

- (Windows) mod_f4fhttp.so, adbe_F4V.dll, libexpat.dll
- (Linux) mod_f4fhttp.so, libF4V.so, libexpat.so

## Configure on-demand streaming

1 Open the Apache \conf\httpd.conf file in a text editor.

2 Locate the LoadModule section and add the following line:

```
LoadModule f4fhttp_module modules/mod_f4fhttp.so
```

3 Configure the HTTP Origin Module to process HTTP requests that are targeted to a sandbox location. The module is not allowed to access files outside the sandbox.

Add the following:

```
<Location /vod>
    HttpStreamingEnabled true
    HttpStreamingContentPath "C:\Program Files\Apache Software
Foundation\Apache2.2\htdocs\vod"
</Location>
```

Substitute your own values for the values in italics.

| Directive | Description |
|---|---|
| `<Location>` | The section of the request URL that maps to the physical location you specify in the `HttpStreamingContentPath` directive. Do not include an http prefix (such as "http://servername") in the `<Location>` path. |
| `HttpStreamingEnabled` | Indicates whether HTTP streaming is enabled (`true`) or not (`false`). |
| `HttpStreamingContentPath` | The physical location of the content. Use an absolute path that includes the DocumentRoot.<br><br>In this example, the HTTP Origin Module can process files in the C:\Program Files\Apache Software Foundation\Apache2.2\htdocs\vod folder or in any of its subfolders. |

4 Save and close the file.

5 Restart Apache HTTP Server.

***Note:** An httpd.conf file can contain multiple `Location` directives. You can configure a `Location` directive for on-demand media and another for live media.*

## Requesting on-demand files

In the httpd.conf file `HttpStreamingContentPath` directive, configure a location for physical files on the server. To request a file, a video player uses the `Location` path + the additional path to the file.

Suppose the following is the `Location` configuration:

```
<Location /vod>
    HttpStreamingEnabled true
    HttpStreamingContentPath "C:\Program Files\Apache Software
Foundation\Apache2.2\htdocs\vod"
</Location>
```

This configuration processes any request that contains the path /vod. The HTTP Origin Module maps the request to the `HttpStreamingContentPath`, for example:

| Request URL | Physical file served to client |
|---|---|
| http://example.com/vod/sample.f4m | C:\Program Files\Apache Software Foundation\Apache2.2\htdocs\vod\sample.f4m |
| http://example.com/vod/free_content/demo.f4m | C:\Program Files\Apache Software Foundation\Apache2.2\htdocs\vod\free_content\demo.f4m |
| http://example.com/index.html | Not processed. |

### More Help topics

"Tutorial: On-demand HTTP Dynamic Streaming" on page 6

"Using the OSMF Player" on page 5

## Configure live streaming

1 Open the Apache \conf\httpd.conf file in a text editor.

2 Locate the LoadModule section and add the following line:

```
LoadModule f4fhttp_module modules/mod_f4fhttp.so
```

3 Configure the HTTP Origin Module to process HTTP requests that are targeted to a sandbox location. The module is not allowed to access files outside the sandbox.

Add the following:

```
<Location /live>
    HttpStreamingEnabled true
    HttpStreamingLiveEventPath "C:\Program Files\Adobe\Flash Media Server 3.8\applications"
    HttpStreamingContentPath "C:\Program Files\Adobe\Flash Media Server 3.8\applications"
</Location>
```

Substitute your own values for the values in italics.

| Directive | Description |
|---|---|
| `<Location>` | The section of the request URL that maps to the physical locations in the `HttpStreamingContentPath` directive and in the `HttpStreamingLiveEventPath` directive. Do not include an http prefix (such as "http://servername") in the `<Location>` path. |
| `HttpStreamingEnabled` | Indicates whether HTTP streaming is enabled (`true`) or not (`false`). |
| `HttpStreamingContentPath` | The physical location of the stream files. |
| `HttpStreamingLiveEventPath` | The physical location of the live event and .f4m file. |

4 Save and close the file.

5 Restart Apache HTTP Server.

***Note:*** *An httpd.conf file can contain multiple <Location> directives. You can configure a <Location> directive for on-demand media and another for live media.*

## Requesting live streams

To play content in the OSMF Player, request a manifest file (.f4m). The OSMF Player contains logic that requests additional files when a user seeks, pauses, and so on.

***Important:*** *For live use cases, the manifest file (.f4m) is not physically present on the disk. The HTTP Origin Module generates the manifest file when the file is requested.*

In the httpd.conf file, you configure two locations for physical files, `HttpStreamingContentPath` and `HttpStreamingLiveEventPath`. To request a file, a video player appends the `Location` path with "/streams" or "/events" so the HTTP Origin Module knows which location to use.

Requests to the `Location` path + "/streams" map to the `HttpStreamingContentPath` folder.

Requests to the `Location` path + "/events" map to the `HttpStreamingLiveEventPath` folder.

Suppose the following is the `Location` configuration:

```
<Location /live>
    HttpStreamingEnabled true
    HttpStreamingLiveEventPath "C:\Program Files\Adobe\Flash Media Server 3.8\applications"
    HttpStreamingContentPath "C:\Program Files\Adobe\Flash Media Server 3.8\applications"
</Location>
```

This configuration processes any request that contains the path /live/events or /live/streams, for example:

| Request URL | Physical file served to client |
|---|---|
| http://example.com/live/events/livepkgr/events/_definst_/liveevent/livestream.f4m | C:\Program Files\Adobe\Flash Media Server 3.8\applications\livepkgr\events_definst_\liveevent\livestream.f4m |
| http://example.com/live/streams/livepkgr/streams/_definst_/livestreamSeg1-Frag1 | C:\Program Files\Adobe\Flash Media Server 3.8\applications\livepkgr\streams_definst_\livestreamSeg1.f4f |
| http://example.com/livestream.f4m | Not processed. |

To use the `NetStream.appendBytes()` API to build an HTTP Dynamic Streaming player, you need to request additional files. The following are the URL request formats when Apache is configured for HTTP Dynamic Streaming:

| Request type | URL Form |
|---|---|
| Manifest file (.f4m) | http://<host>/<location-tag-alias>/events/<app-name>/event/<app-instance>/<event-name>.f4m<br><br>The <location-tag-alias> is the path in the Location directive in the httpd.conf file. |
| Fragment | http://<host>/<location-tag-alias>/streams/<app-name>/streams/<app-instance>/<stream name>Seg<segment #>-Frag<fragment #><br><br>**Note:***You cannot request an F4F file directly. Request a Fragment.* |
| Bootstrap (.bootstrap) | http://<host>/<location-tag-alias>/streams/<app-name>/streams/<app-instance>/<stream name>.bootstrap |
| Manifest file (.f4m) when an Event.xml file is not configured.<br><br>Retrieves a default manifest for playback of a single stream not associated with an event. | http://<host>/<location-tag-alias>/events/defevent.f4m |

### More Help topics

"Tutorial: Live HTTP Dynamic Streaming" on page 9

"Using the OSMF Player" on page 5

# Logging

## Access log

All HTTP access requests are logged in the Apache access_log file in the \logs directory. The HTTP Origin Module doesn't add any logging directives. All access requests are logged through the standard Apache HTTP Server log module.

## Error log

All errors are logged in the Apache error_log file in the \logs directory. The following table shows the response codes that the HTTP Origin Module returns.

| Response code | Error message | Description |
|---|---|---|
| 200 | None. | No error occurs, and a response has been returned successfully. |
| 304 | None | The HTTP Origin Module supports "If-Modified-Since" in the request header and returns 304 if the file has not been modified."If-Modified-Since" is not support in requests for manifest files of live events. |
| 400 | mod_f4fhttp [400]: Syntax error in %s | Error occurs while parsing the URI request. The variable %s is the URI string. For example, the request URI http://servername/media/sample- is missing a fragment number. |

| Response code | Error message | Description |
|---|---|---|
| 403 | mod_f4fhttp [403]: No access to %s<br><br>mod_f4fhttp [403]: Internal error %1 (%2) when processing %3 | Error occurs when accessing the file. The variable %s is the file path. For example, the request URl is http://servername/media/sample.fmf and the module doesn't have permission to read the file sample.f4m. Error occurs when the library fails to access a file. The variable %1 is the error code from the library, %2 is error description, and %3 is the url string. |
| 404 | mod_f4fhttp [404]: %s does not exist<br><br>mod_f4fhttp [404]: Internal error %1 (%2) when processing %3 | Error occurs when the request file is not found. The variable %s is the file path. For example, the request URl is http://servername/media/sample.fmf but the file sample.f4m does not exist.<br><br>Error occurs when the library fails to find a file. The variable %1 is the error code from the library, %2 is the error description, and %3 is the url string. |
| 500 | mod_f4fhttp [500]: Unknown exception when processing %1<br><br>mod_f4fhttp [500]: Internal error %1 (%2) when processing %3 | Unknown error occurs when processing a request, where %1 is the url string.<br><br>Example: http://servername/media/sampleSeg1-Frag1, sampleSeg1.f4f is not a corrupted file. |
| 503 | None. | The HTTP Origin Module returns a 503 code so the client and the proxy servers don't cache the response.<br><br>In a live event, the file can be growing and a fragment could be missing in one origin server but not missing on another origin server. The HTTP response header includes an "X-mod_f4fhttp" field to provide more details. The following is a sample response header:<br><br>Date Thu, 04 Mar 2010 06:20:20 GMT Server Apache/2.2.9 (Win32) Content-Length 432 Last-Modified Wed, 27 Jan 2010 11:43:25 GMT Keep-Alive timeout=5, max=100 Connection Keep-Alive Content-Type video/f4f X-mod_f4fhttp 1, error 71<br><br>There are two parameters in the X-mod_f4fhttp field delimited by a comma. The first parameter is a status code and the second parameter is the error code from the library.<br><br>There are two status codes: 1 - The requested fragment number is larger than the last fragment in the file (the corresponding error is 71). 2 - The requested fragment number is a missing fragment (the corresponding error is 34). |

# Chapter 3: Packaging on-demand media

The File Packager is a command-line tool that packages on-demand media files into a format that can stream over HTTP. The File Packager can also encrypt files for protection with Flash Access.

## Packaging on-demand media

### File Packager location

The File Packager is available from the following locations:

- adobe.com/products/httpdynamicstreaming
- Flash Media Interactive Server *rootinstall*/tools/f4fpackager folder

The File Packager requires several libraries to run. These libraries are located in the f4fpackager folder. To move the File Packager, move the f4fpackager folder and all its contents.

### Input file formats and output file formats

To stream on-demand media over HTTP, process the file with the File Packager. The File Packager parses the content, translates it into segments and fragments, and writes the segments and fragments to the F4F format. You can also choose to have the File Packager encrypt the content for use with Flash Access.

The File Packager supports the following file formats:

- F4V/MP4 compatible files
- FLV

***Important:*** *The file formats can contain any codec the format supports, except Speex audio. You can package a file that includes Speex audio, the File Packager does not throw an error, but the audio will not play.*

The File Packager outputs the following files:

| File type | Description |
|---|---|
| .f4f | A segment. The tool outputs one or more F4F files. Each file contains a segment of the source file. Each segment contains one or more fragments of content. A player can use a URL to address each fragment. |
| .f4m | Flash Media Manifest file. Contains information about codec, resolution, and the availability of multi-bitrate files. |
| .f4x | Index file. Contains the location of specific fragments within a stream. |
| .bootstrap | Bootstrap file. Contains the bootstrap information for each segment of the file. The tool creates an external .bootstrap file only if you specify the `--external-bootstrap` option. If you don't specify that option, the bootstrap information is included in the manifest file. |
| .drmmeta | DRM Additional Header file. Contains additional header information about an encrypted file. The tool creates this file only if you specify the `--external-bootstrap` option and encrypt the file. Otherwise, the information is included in the manifest file. |

## Segments And Fragments

An F4f file contains one segment of a media file. Each segment can contain multiple fragments. Use the `--segment-duration` and `--fragment-duration` options to set the length of each segment and fragment, in seconds. A good size fragment is small enough to send over the internet and long enough to prevent multiple calls to Apache HTTP Server.

## Run The File Packager

To package a file, pass the File Packager the name of the input file.

There are two ways to specify the name:

- Enter the `--input-file` option at the command line.
- Create a configuration file that specifies the input file and enter the `--conf-file` option at the command line.

### More Help topics

"Command-line options to fragment files" on page 24

"Command-line options to encrypt files" on page 26

"File Packager configuration file" on page 27

## Package content on Windows

1. Open a Command Window.
2. Change directories to the directory containing f4fpackager.exe.
3. Enter the name of the tool and any options:

   ```
   f4fpackager --input-file=sample.f4v --output-path=c:\sampleoutput
   ```

   The following files are output: sampleSeg1.f4f, sample.f4x, and sample.f4m.

### More Help topics

"Command-line options to fragment files" on page 24

"Command-line options to encrypt files" on page 26

"File Packager configuration file" on page 27

## Package content on Linux

1. Open a terminal window.
2. On a clean installation of Linux, set the `LD_LIBRARY_PATH` to the directory containing the File Packager libraries.
3. At the shell prompt, enter the name of the tool and any options:

   ```
   f4fpackager --input-file=sample.f4v --output-path=/sampleoutput
   ```

   The following files are output: sampleSeg1.f4f, sample.f4x, and sample.f4m.

### More Help topics

"Command-line options to fragment files" on page 24

"Command-line options to encrypt files" on page 26

"File Packager configuration file" on page 27

## Packaging multi-bitrate content

To stream multi-bitrate content, encode a piece of media at multiple bitrates, creating multiple files. Use the File Packager to process each media file individually. The media files share a manifest file that lists information about each media file. The OSMF Player uses this information for playback.

### More Help topics

"Command-line options to fragment files" on page 24

"Command-line options to encrypt files" on page 26

"File Packager configuration file" on page 27

### About packaging multi-bitrate files

Each time you run the File Packager, it creates a manifest file with the filename of the input file. When you pass the `--manifest-file` option, the File Packager adds the information from the manifest file you specify to the manifest file it creates. For example, suppose you run the following:

```
--input-file=sample1.f4v
```

The tool creates the files: sample1Seg1.f4f, sample1.f4x, and sample1.f4m.

Suppose you then run the following:

```
--input-file=sample2.f4v --manifest-file=sample1.f4m
```

The tool creates the files sample2Seg1.f4f, sample2.f4x, and sample2.f4m. The manifest file sample2.f4m contains information about sample2.f4v and sample1.f4v.

To create a manifest file for multi-bitrate content, pass the `--manifest-file` option and specify the manifest file of the previously processed file. Package the files from lowest bitrate to highest bitrate. Do not pass the `--manifest-file` option for the first file you process.

### Package multi-bitrate files

***Note:*** *This example uses Windows. For Linux commands, see "Package content on Linux" on page 21.*

1. Encode a media file at several bitrates, for example, 150 kbps, 700 kbps, and 1500 kbps.

   This example uses three files: sample1_150kbps.f4v, sample1_700kbps.f4v, and sample1_1500kbps.f4v. On Flash Media Server, these files are installed to *rootinstall*\applications\vod\media.

2. Copy the files to the same folder as the File Packager.

   On Flash Media Interactive Server 3.8, the File Packager is located in the *rootinstall*\tools\f4fpackager folder.

3. Open a Command Window.
4. Change directories to the directory containing f4fpackager.exe.
5. Enter the following:

   ```
   f4fpackager --input-file=sample1_150kbps.f4v --bitrate=150
   ```

The following files are output: sample1_150kbpsSeg1.f4f, sample1_150kbps.f4x, and sample1_150kbps.f4m.

6 Enter the following to process the second file:

```
f4fpackager --input-file=sample1_700kbps.f4v --manifest-file=sample1_150kbps.f4m --
bitrate=700
```

The following files are output: sample1_700kbpsSeg1.f4f, sample1_700kbps.f4x, and sample1_700kbps.f4m. The manifest file sample1_700kbps.f4m includes information from the input file and information from the sample1_150kbps.f4m manifest file.

7 Enter the following to process the third file:

```
f4fpackager --input-file=sample1_1500kbps.f4v --manifest-file=sample1_700kbps.f4m --
bitrate=1500
```

The following files are output: sample1_1500kbpsSeg1.f4f, sample1_1500kbps.f4x, and sample1_1500kbps.f4m. The manifest file sample1_1500kbps.f4m includes information from the input file and information from the sample1_700kbps.f4m manifest file. The sample1_700kbps.f4m file contains information about the sample1_150kbps.f4v file and the sample1_700kbps.f4v file.

8 To play the file, pass the URL of the sample1_1500kbps.f4m file to OSMF Player. Because this manifest file was created last, it contains information about all three files.

## Package and encrypt content

Many command-line options are required to encrypt a file. It's easier to use a configuration file to set options than it is to set options at the command line.

1 Open the file f4fpackager_config.xml in a text editor.

By default, the tool looks for f4fpackager_config.xml in the same directory as the File Packager. However, you can move it to any directory and give it any name.

2 Enter values for the following required options (sample values are included here):

```
<offline>
    <input-file>someFile.f4v</input-file>
    <content-id>contentId</content-id>
    <common-key>commonKey.bin</common-key>
    <license-server-url>http://server1.com:9999</license-server-url>
    <license-server-cert>licenseServer.der</license-server-cert>
    <transport-cert>transportCert.der</transport-cert>
    <packager-credential>packagerCredential.pfx</packager-credential>
    <credential-pwd>mYpwd</credential-pwd>
    <policy-file>policyFile.pol</policy-file>
</offline>
```

You can add additional elements to change their default values.

3 Save the configuration file.

4 Open a Command Window.

5 Change directories to the directory containing f4fpackager.exe.

6 Enter the following:

```
f4fpackager --conf-file=f4fpackager_config.xml
```

### More Help topics

"Command-line options to fragment files" on page 24

"Command-line options to encrypt files" on page 26

"File Packager configuration file" on page 27

# File Packager reference

## Command-line options to fragment files

You can use the command line to set all the options for the File Packager. Options set at the command line override any options set in the configuration file. The only required option is `--input-file`.

| Option | Description | Required | Default value |
|---|---|---|---|
| --input-file | The path to the source container file. The path can be absolute or relative to the directory containing the File Packager.<br><br>If you specify the `--conf-file` option and the configuration file contains an <input-file> value, you do not need to use the `--input-file` option. | Yes | None |
| --output-path | The path to the directory to which the packaged files are output. If the directory doesn't exist, the tool creates it. The path can be absolute or relative to the directory containing the File Packager. | No | Current directory |
| --segment-duration | The length, in seconds, of a segment. The default value is 0 which creates 1 segment. | No | 0 |
| --fragment-duration | The target length of each fragment, in seconds. The actual length of each fragment depends on properties of the input file, such as the keyframe interval. | No | 4 |
| --duration-precision | How precise the fragment durations are, in seconds. When using a time-based fragment duration, duration precision controls how much rounding is applied to the fragment durations in the fragment run tables. This prevents small variations in the fragment durations from causing the run tables to grow too large. | No | 0.1 |
| --frame-rate | The frame rate of the original content, in frames per second (fps). The value is floating point; for NTSC, use the value 29.97. | No | |
| --frame-precision | This option is the frame-based equivalent to the --duration-precision option. This value is the number of frames to round to in the fragment run tables. | No | 1 |
| --frames-per-keyframe-interval | The number of frames between each keyframe. For example, 30 fps video with a keyframe every 2 seconds contains 60 frames per keyframe interval. | No | |

| Option | Description | Required | Default value |
|---|---|---|---|
| --keyframe-intervals-per-fragment | The number of keyframe intervals per fragment. The default value is 1 which sets the fragment size to the same size as the keyframe interval. | No | 1 |
| --allowed-drift | As the tool rounds off durations, it drifts from the real time. This option specifies how far the tool can vary from the real time, in seconds. | No | 1 |
| --conf-file | The configuration file that contains settings for the packaging process.<br>You can specify an absolute path or a path relative to the directory containing the tool. You can use the filename f4fpackager_config.xml or any other legal filename.<br>Suppose the tool is in the c:\media directory and the configuration file is in the c:\media\test directory. Run the following from the command line: `f4fpackager --conf-file=test\f4fpackager_config.xml`. | No | f4fpackager_config.xml<br>If the f4fpackager_config.xml file is not in the current directory, the default value is "none". |
| --manifest-file | Updates an existing manifest file. Use this option to package a file encoded at multiple bitrates for dynamic streaming. For example, you could package three files (sample_500kpbs.f4v, sample_1000kbps.f4v, and sample_1500kbps.f4v) and add them all to the same manifest file.<br>If you omit this option, the tool creates a Flash Media Manifest file (F4M). | No | None |
| --bitrate | Specifies the bitrate of the content to fragment and adds it to the manifest file. Use this option to serve files encoded at multiple bitrates. Use this option or manually edit the manifest file to add the bitrate. | No | 0 |
| --external-bootstrap | Outputs the bootstrap box to a .bootstrap file. If you omit this option, the information in the bootstrap box is placed in the manifest file (F4M).<br>Outputs DRM additional header information to a .drmmeta file when a file is encrypted. If you omit this option, the additional header information is included in the manifest file. | No | None |
| --config-dump | Dumps the configuration settings. This option does not fragment the file. | No | None |
| --sample-dump | Dumps the samples and the fragments they're set to. This option does not fragment the file. | No | None |
| --progress | Prints progress updates. | No | None |
| --help | Lists and describes every File Packager command. | No | None |

## Setting fragment duration based on frames or time

The File Packager has options that determine the size of fragments. There is a frame-based option, `--keyframe-interval-per-fragment`, and a time-based option, `--fragment-duration`. Each option has additional options that refine how the tool creates fragments.

Use the frame-based option when the source media contains video encoded at a constant frame rate. The frame-based option lets you carefully match the fragment size to the video's keyframe interval. Use the time-based option for media that contains audio or data but not video.

Frame-based options: `--keyframe-interval-per-fragment`, `--frames-per-keyframe-interval`, `--frame-rate`, `--frame-precision`.Time-based options: `--fragment-duration`, `--duration-precision`. The frame-based option overrides the time-based option. If you don't provide a `--keyframe-interval-per-fragment` option, the File Packager uses the `--fragment-duration` option. If you don't provide values for `--keyframe-interval-per-fragment` or `--fragment-duration`, the File Packager uses the default fragment duration, which is 4 seconds.

## Command-line options to encrypt files

Use the File Packager to encrypt files for use with Flash Access 2.0. Pass digital certificates for some of the following options. To obtain digital certificates, see the Flash Access Certificate Enrollment Site.

| Option | Description | Required | Default |
|---|---|---|---|
| --transport-cert | The DER encoded transport certificate file. | Yes | None |
| --license-server-url | The URL of the license server that handles license acquisition for this content. | Yes | None |
| --license-server-cert | The DER encoded license server certificate file used for content protection. | Yes | None |
| --packager-credential | The PFX file containing the packager's protection credentials. | Yes | None |
| --credential-pwd | The password string used to secure the packager credentials. | Yes | None |
| --common-key | A file containing the base key. The base key is used with the content ID to generate the content encryption key. The base key must be randomly generated and 16 bytes.<br>For dynamic streaming (also called multi-bitrate streaming), use the same common key and content ID for an entire set of content. Using the same key and id allows a single license to decrypt a set of content. | Yes | None |
| --content-id | The content ID used with the common key to generate the content encryption key. | Yes | None |
| --policy-file | The file containing the policy for this content. Currently only a single policy can be applied to content packaged with this tool. | Yes | None |
| --encrypt-audio | Specifies whether to encrypt the audio in the file. | No | true |
| --encrypt-video | Specifies whether to encrypt the video in the file. | No | true |

| Option | Description | Required | Default |
|---|---|---|---|
| --encrypt-data | Specifies whether to encrypt the data in the file. This option supports both FLV and F4V/MP4 formats. | No | true |
| --ms-unencrypted | The number of milliseconds at the beginning of the content that remains unencrypted. Use this option to improve stream start times because it allows the client to acquire the license asynchronously as playback begins. | No | 0 |
| --video-encrypt-level | The level of encryption for H.264 content. Possible values are 0 (low), 1(medium), and 2 (high). The default value is 2.<br>**Note:** *Using this option on non-H.264 content won't generate an error, but it won't change the encryption level.*<br>The values "0" and "1" mean "partial encryption"; only important samples like video keframes are encrypted. This setting creates fewer frames to decrypt. Use this setting to improve playback performance on the client.<br>The value "2", encrypts all video samples (after --ms-unencrypted, if specified).. | No | 2 |

## File Packager configuration file

The File Packager uses an XML configuration file to read values for command-line options. The default configuration file, f4fpackager_config.xml, is located in the same directory as the File Packager. It is often easier to specify options in a text file that you can reuse than it is to enter options at the command line.

Options you set at the command line override options set in the configuration file. This feature lets you create a configuration file for a set of content but edit a few settings for individual files.

***Note:*** *You do not need to use the configuration file to run the tool.*

Enter the `--conf-file` option at the command line to pass the name and location of a configuration file. If you don't enter the `--conf-file` option, the tool checks for a file called f4fpackager_config.xml file in the same directory as the tool.

You can enter all the options for packaging and encrypting, including the input file, in the configuration file.

The following is the format of the configuration file:

```
<offline>
    <input-file>/media/input-file.f4v</input-file>
    <output-path>/media/http</output-path>
    <fragment-duration></fragment-duration>
   ... You can set as many options as you want ...
</offline>
```

For options that take a path, you can specify absolute paths or paths relative to the directory containing the tool.

## Errors

The File Packager outputs the following errors:

| Error code | Error message | Description |
|---|---|---|
| 1 | File not found | The input file, the manifest file or the configuration file could not be found. |
| 2 | Input File not given | No input file has been specified. To run the File Packager, you must specify a path to a supported file type. See "Input file formats and output file formats" on page 20. |
| 3 | Invalid input file | The input file, manifest file, or configuration file is an invalid input file. |
| 4 | Can't access file | The input file, manifest file, or configuration file cannot be accessed. |
| 5 | Unsupported codec | The input file contains unsupported codecs. |
| 6 | No supported tracks | The input file doesn't contain any supported tracks. |
| 7 | Write error | A write error occurred. Verify that the input file is structured correctly and that there is enough memory to run the File Packager. |
| 8 | XML library error | An error occur ed when parsing the configuration file. Verify that the XML is structured correctly. |
| 9 | Program Options error | The program options library failed. Verify that the command line arguments are structured correctly. |
| 10 | Offline Encryption info error | The File Packager doesn't have enough information to encrypt the file. Verify that there aren't any missing inputs. |
| 11 | Encryption library error | The encryption library threw an error while encrypting the content. |
| 12 | Standard error | A Standard Exception has occur ed. Verify that there is enough memory to run the File Packager. |
| 13 | Invalid Bootstrap error | The bootstrap file is corrupt and cannot be parsed. |
| 14 | FLV Error | The FLV file cannot be read. |
| 999 | Internal error | There has been an internal error in the File Packager. |
| 1000 | Invalid Input file warning | Invalid input file warning. |
| 2000 | Invalid Error code | The error code is invalid. |

# Chapter 4: Packaging live media

Use Flash Media Interactive Server to package and deliver live streams to Flash Player over HTTP. Live HTTP Dynamic Streaming supports adaptive streaming, DVR functionality, and Flash Access protection.

Flash Media Interactive Server 3.8 and later include the Live Packager for HTTP Dynamic Streaming, the File Packager (for packaging on-demand content), and a version of the HTTP Origin Module that serves live and on-demand content. The server also installs with the livepkgr application which supports live HTTP Dynamic Streaming.

## Configuring the server for live packaging

### About live events

***Important:*** *Flash Media Server installs with an application called "livepkg". The livepkgr application contains a live event called "liveevent". To see the configuration, browse to the folder rootinstall/applications/livepkgr/events/_definst_/liveevent.*

To stream live media over HTTP, Flash Media Server introduces the concept of *live events*. A live event is a configuration level within an application. You can have multiple live events within one application. Use a live event to add metadata about multi-bitrate streaming, DVR, and Flash Access encryption to a set of incoming streams.

You must create a live event to use any of the following features:

- Multi-bitrate streaming
- DVR
- Flash Access protection

***Note:*** *To serve a single, unencrypted, live stream you don't have to use a live event.*

Like a Flash Media Server application, a live event is a folder on the server. Like a Flash Media Server application, the folder name is the name of the live event. The following is the location of the live event:

*rootinstall*/applications/*applicationname*/events/*applicationinstancename*/*liveeventname*

By default, Flash Media Server includes an application called "livepkgr". This application contains an event called "liveevent":

*rootinstall*/applications/livepkgr/events/_definst_/liveevent

***Note:*** *The "events" folder is a sibling to the "streams" folder that the server uses by default for on-demand content and recorded live content.*

When using HTTP Dynamic Streaming, you can configure the location of the "applications" folder. You can also configure virtual directory mappings for the "streams" folder. However, you cannot configure virtual directory mappings for the "events" folders.

Streams in a live event are packaged as fragments and written to disk. A live event (the stream content and the metadata) exists until you delete it. A player can access the content after the source stream has stopped publishing, like a DVR.

## Create a live event

1 Create the following "events" folder structure in the application folder:

*rootinstall*/applications/*applicationname*/events/*applicationinstancename*/*liveeventname*

For example, the live event "liveevent" lives inside the default instance of the livepkgr application:

*rootinstall*/applications/livepkgr/events/_definst_/liveevent

2 Open a text editor and create an Events.xml file. Save the file to the live event folder (in this example, liveevent).

The Events.xml file contains fragment and segment length, and DRM additional header metadata. To package and encrypt content for use with Flash Access, configure an Events.xml file. See "Event.xml file" on page 34.

3 To create a DVR or multi-bitrate live event, open a text editor and create a Manifest.xml file. Save the file to the live event folder.

The Manifest.xml file contains DVR and multi-bitrate settings. See "Manifest.xml file" on page 36.

### More Help topics

"Live workflow" on page 4

"Tutorial: Live HTTP Dynamic Streaming" on page 9

## Publish a stream to the Live Packager and associate the stream with a live event

***Important:*** *Flash Media Server installs with an application called "livepkg". The livepkgr application contains a main.asc file that publishes streams to the Live Packager and associates streams with a live event. To see the code, open rootinstall/applications/livepkgr/main.asc in a text editor.*

To publish a stream to the Live Packager, specify `f4f:` before the stream name. Flash Media Live Encoder doesn't support the `f4f:` prefix so add it to the stream name in a server-side script or in a Flash Media Server Authorization Plug-in.

An application can contain more than one live event. For that reason, a stream published to an application is not associated with a live event by default. Use a server-side script or the Flash Media Server Authorization Plug-in (not both) to associate a live stream with a live event. You can associate a live stream with only one live event.

***Note:*** *To serve a single, unencrypted, live stream published from a custom client (not from Flash Media Live Encoder), you don't have to associate the live stream with a live event.*

Associate a stream during the publish event. When the stream is published and associated, the server creates a stream record file (.stream) for the stream in the event directory:

applications/*appname*/events/*appinstancename*/*liveeventname*/*livestream*.stream

The stream record file contains information about the location of the packaged stream files. The HTTP Origin Module uses this information to generate an .f4m manifest file. For more information, see "Stream record files" on page 40.

### Use Flash Media Live Encoder and Server-Side ActionScript

1 Publish a stream from Flash Media Live Encoder.

Do not specify the "f4f:" prefix in the Stream name. Flash Media Live Encoder does not support the "f4f:" prefix. Do not use the Record button, the DVR Auto Record setting, or the Save to File setting. To record the stream on the server, you must use the Server-Side ActionScript Stream class.

2 In the Server-Side ActionScript code, use the `application.onPublish` event to do the following:

- Use the `Stream.liveEvent` property to associate a live stream with live event.

  Associate the livestream with an event before the server starts recording.

- Add the `f4f:` prefix to the stream name. This tells the server to package the stream.
- Call `Stream.record()` to record the stream.

## Use Flash Media Live Encoder and the Flash Media Server Authorization Plug-in

1 Publish a stream from Flash Media Live Encoder.

Do not specify the "f4f:" prefix in the Stream name. Flash Media Live Encoder does not support the "f4f:" prefix. Do not use the Record button, the DVR Auto Record setting, or the Save to File setting. To record the stream on the server, you must use the Server-Side ActionScript Stream class.

2 In the Authorization Plug-in code, use the `E_PUBLISH` event to do the following:

- Set the `F_STREAM_TYPE` field to `"f4f"`.
- Set the `F_STREAM_LIVE_EVENT` field to *`"liveeventname"`*. This example uses "liveevent".
- Set the `F_STREAM_PUBLISH_TYPE` field to `0` which means "record".

The following is sample C++ code that handles the `E_PUBLISH` event:

```
case IFmsAuthEvent::E_PUBLISH:
{
    // The name of the FMS app to which the live stream is published.
    char* pLiveApp = "livepkgr";
    // The stream type used for HTTP Dynamic Streaming.
    char* pStreamType = "f4f";
    // The name of the live event defined in the livepkgr app.
    char* pLiveEvent = "liveevent";
    // The Auth Plug-in affects all apps on the server.
    // We only want to process streams published to the livehttp app.
    char* pAppName = getStringField(m_pAev, IFmsAuthEvent::F_APP_NAME);
    if (pAppName && !strncmp(pAppName, pLiveApp, strlen(pLiveApp)))
    {
        // Set the stream type.
        setStringField(m_pAev, IFmsAuthEvent::F_STREAM_TYPE, pStreamType);
        // Set the publish type to record.
        //0 record, 1 append, -1 live
        setI32Field(m_pAev, IFmsAuthEvent::F_STREAM_PUBLISH_TYPE, 0);
        // Associate the stream with a live event.
        setStringField(m_pAev, IFmsAuthEvent::F_STREAM_LIVE_EVENT, pLiveEvent);
    }
}
```

## Use a custom publishing client

When you publish a stream from a custom client, you can add the "f4f:" prefix and tell the server to record the stream in the call to `NetStream.publish()`. For example, the following code publishes a stream called "livestream" and tells the server to fragment it and record it:

```
myNetStream.publish("f4f:livestream", "record");
```

Use a server-side script or the Authorization Plug-in to associate the stream with a live event. In a server-side script, use the `Stream.liveEvent` property. In the Authorization Plug-in, use the `F_STREAM_LIVE_EVENT` field.

### Match fragment duration and keyframe interval

The server records the ingested stream into fragments. On the server, the fragment duration is set in the Event.xml file. The default fragment duration is 4000 milliseconds (4 seconds).

The server's fragment duration and the encoder's keyframe interval must be the same. For example, to use the default fragment duration, configure the encoder to use a 4 second keyframe interval. To use a different keyframe interval, change the `FragmentDuration` value in the Event.xml file to match the encoder setting.

To set keyframe intervals in Flash Media Live Encoder, click the wrench beside the Format in the Encoding Options panel. In the Advanced Encoding Options window, select 4 seconds from the Keyframe Frequency pop-up menu.

To set keyframe intervals in a custom Flash Player client, call `Camera.setKeyFrameInterval()`.

### Configure absolute time for the live encoder

To publish multiple streams encoded at different bit rates (called *dynamic streaming*, *adaptive streaming*, and *multi-bitrate streaming*), the timeline of the live streams must use absolute time.

Each live encoder supports absolute time differently. Refer to the documentation for your live encoder.

To configure absolute time for Flash Media Live Encoder, do the following:

1. Close Flash Media Live Encoder.
2. Open the Flash Media Live Encoder the *rootinstall*\Conf\config.xml file in a text editor.
3. Set the tag //flashmedialiveencoder_config/mbrconfig/streamsynchronization/enable to true:

```
<flashmedialiveencoder_config>
    <mbrconfig>
        <streamsynchronization>
            <!-- "true" to enable this feature, "false" to disable.-->
            <enable>true</enable>
```

4. Save the file.

## Configure absolute time on the server

***Important:*** *Flash Media Server installs with an application called "livepkgr" that is configured for absolute time. To see the configuration, open rootinstall/applications/livepkgr/Application.xml in a text editor.*

To publish live, multi-bitrate streams, configure the live encoder to publish the streams using absolute time.

When you publish a stream that uses absolute time, configure the ingesting application on the server to use absolute time. With this configuration, the server assumes that all live streams published to the application use absolute time.

1. Copy an Application.xml file to the application folder.

   For example, if you're publishing streams to the application "livehttp", copy an Application.xml file from the conf_defaultRoot__defaultVHost_ folder to the applications/livehttp folder.

2. Open the Application.xml file in a text editor.
3. Set the tag //Application/StreamManager/Live/AssumeAbsoluteTime to true:

```
<Application>
    ...
    <StreamManager>
        ...
        <Live>
            ...
            <AssumeAbsoluteTime>true</AssumeAbsoluteTime>
```

4 Save the file.

5 Restart the server. See Starting and stopping the server.

***Important:*** *Set <AssumeAbsoluteTime> to true only when all streams published to the application use absolute time. If this value is set to true and the application ingests a stream that does not use absolute time, you may see warnings in the server logs about time going backwards when streams are stopped and re-published.*

## Configure the recording buffer on the server

1 Open the Server.xml configuration file in a text editor. The Server.xml file is located at *rootinstall*/conf.

2 Locate the <MaxFlushTime> element:

```
<Root>
    <Server>
        ...
        <ResourceLimits>
            <RecBuffer>
                <MaxFlushTime>
```

3 Set the value of <MaxFlushTime> to 1.

The <MaxFlushTime> is the maximum number of seconds the server waits before flushing the record buffer. Lower values enable users to watch the recording with a lower latency relative to live, but at the cost of higher disk usage. The default value is 5 seconds. The minimum value is 1 second

If <MaxFlushTime> in the Server.xml file is longer than <FragmentDuration> set in the Event.xml file, it can introduce latency writing F4F fragments to the disk. Adobe recommends setting <MaxFlushTime> to 1 second (the minimum) for HTTP Dynamic Streaming.

4 Save the Server.xml file.

5 Restart Flash Media Server.

See Starting and stopping the server.

# Live Packager reference

## Input codecs

The Live Packager for HTTP Dynamic Streaming ingests live streams and writes them to F4F files. The Live Packager supports all the codecs that Flash Player supports, except Speex audio.

## Output file types

When you publish in F4F file format, the server creates the following files in the *applicationname*/streams/*instancename* folder:

| File | Description |
|---|---|
| .f4f | A segment. Flash Media Interactive Server 3.8 outputs one or more F4F files. Each file contains a segment of the source file. Each segment contains the fragmented (and optionally protected) content. |
| .f4x | An index file listing the fragment offsets in each .f4f file. Flash Media Interactive Server 3.8 outputs one or more F4X files. The HTTP Origin Module uses this file to deliver fragments. |
| .meta | Contains the stream metadata (bitrate, screen size, and so on). |
| .bootstrap | Contains the bootstrap information for the stream. |
| .drmmeta | Contains the DRM additional header information when a stream is encrypted for use with Flash Access. |
| .control | Contains internal metadata that the Live Packager uses to manage stream state. |

Suppose you publish a stream called "livestream" to the default instance of the livepkgr application. The server creates the following files in the folder applications/livepkgr/streams/_definst_/livestream:

- livestreamSeg#.f4f
- livestream.f4x
- livestream.meta
- livestream.bootstrap
- livestream.drmmeta (if the stream is configured for encryption)

***Important:*** *The HTTP Origin Module generates the .f4m file when it is requested. The file is not physically present on the disk. However, you can request the file even when a live stream has stopped publishing.*

### More Help topics

"Publish a stream to the Live Packager and associate the stream with a live event" on page 30

## Event.xml file

To protect a file for use with Flash Access, configure an Event.xml file.

To set the duration of segments and fragments, configure an Event.xml file.

Copy the Event.xml file to the live event folder.

The following is a sample Events.xml file:

```
<Event>
    <EventID>liveevent</EventID>
    <Recording>
        <FragmentDuration>4000</FragmentDuration>
        <SegmentDuration>10000</SegmentDuration>
        <ContentProtection enabled="true">
            <ProtectionScheme>FlashAccessV2</ProtectionScheme>
            <FlashAccessV2>
                <ContentID>foo</ContentID>
                <CommonKeyFile>common-key.bin</CommonKeyFile>
                <LicenseServerURL>http://dill.corp.adobe.com:8090</LicenseServerURL>
                <TransportCertFile>production_transport.der</TransportCertFile>
                <LicenseServerCertFile>license_server.der</LicenseServerCertFile>
                <PackagerCredentialFile>production_packager.pfx</PackagerCredentialFile>
                <PackagerCredentialPassword>hbXX5omIhzI=</PackagerCredentialPassword>
                <PolicyFile>policy01.pol</PolicyFile>
            </FlashAccessV2>
        </ContentProtection>
    </Recording>
</Event>
```

| Element | Attributes | Description |
|---|---|---|
| Event | | The root element for an Event.xml configuration file. |
| EventID | | The name of the event. This is the same as the name of the event folder and the value of the Server-Side ActionScript `Stream.liveEvent` property. |
| Recording | | The section that configures how the file is written to disk. |
| FragmentDuration | | The length of each fragment, in milliseconds. Each segment can contain one or more fragments. |
| SegmentDuration | | The length of each segment, in milliseconds. Each .f4f file contains one segment. |
| ContentProtectionEnabled | enabled | Whether to enable content for protection with Flash Access. |
| ProtectionScheme | | The protection scheme. This value is always “FlashAccessV2”. |
| FlashAccessV2 | | A container for settings used by Flash Access to protect content. |
| ContentID | | The content ID used with the common key to generate the content encryption key. |
| CommonKeyFile | | The base key used with the content ID to generate the content encryption key. This is a binary file containing a 16 byte binary key.<br>For adaptive streaming, use the same common key and content ID for an entire set of content. Using the same key and id allows a single license to decrypt a set of content. |
| LicenseServerURL | | The URL of the license server that handles license acquisition for this content. |
| TransportCertFile | | The DER encoded transport certificate file. |
| LicenseServerCertFile | | The DER encoded license server certificate file used for content protection. |

| Element | Attributes | Description |
|---|---|---|
| PackagerCredentialFile | | The PFX file containing the packager's protection credentials. |
| PackagerCredentialPassword | | The password string used to secure the packager credentials. |
| PolicyFile | | The file containing the policy for this content. Currently only a single policy can be applied to content packaged with this tool. |

### More Help topics

"Protecting content with Flash Access" on page 5

## Manifest.xml file

To add DVR functionality to a live stream, create a Manifest.xml file with a <dvrInfo> element.

To stream multi-bitrate content, create a Manifest.xml file with a <media> element for each stream.

To use DVR for a single stream, include the <dvrInfo> in the Manifest.xml file and remove the <media> tags.

To use multi-bitrate streaming without DVR, remove the <dvrInfo> tag.

Copy the Manifest.xml file to the live event folder.

The HTTP Origin Module uses the Manfest.xml file to generate a manifest file (.f4m). To play a stream, request the manifest file (.f4m) from OSMF Player.

The following Manifest.xml file demonstrates the DVR and multi-bitrate use cases:

```
<manifest xmlns="http://ns.adobe.com/f4m/1.0">
   <dvrInfo beginOffset="30" endOffset="60"></dvrInfo>
   <media streamId="test1" bitrate="1000">
   </media>
   <media streamId="test2" bitrate="2000">
   </media>
   <media streamId="test3" bitrate="3000">
   </media>
</manifest>
```

| Element | Attributes | Description |
|---|---|---|
| dvrInfo | | |
| | beginOffset | An offset, in seconds, from the beginning of the recorded stream. Clients can begin viewing the stream at this location. The default value is 0.<br>Negative values are treated as 0. It neither `endOffset` nor `beginOffset` is set, the start time is the beginning of the content. |
| | endOffset | An offset, in seconds, behind the current duration of the recorded stream. Clients cannot view the stream behind this location. The default value is 0.<br>Negative values are treated as 0. It neither `endOffset` nor `beginOffset` is set, the start time is the beginning of the content. |

| Element | Attributes | Description |
|---|---|---|
| media | | Represents one stream. |
| | streamId | The name of the publishing stream. |
| | bitrate | The bitrate at which the stream was encoded. |

When the HTTP Origin Module parses the previous sample Manifest.xml file, it generates the following .f4m manifest file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<manifest xmlns="http://ns.adobe.com/f4m/1.0">
    <id>Apps/live/events/definst/myTestEvent</id>
    <mimeType/>
    <streamType>live</streamType>
    <duration>0</duration>
    <dvrInfo beginOffset="30" endOffset="60"></dvrInfo>
    <media
        streamId="test1"
        bitrate="1000"
        url="../../../../../streams/Apps/live/streams/definst/test1/test1"
        bootstrapInfoId="bootstrap7536">
        <metadata>
AgAKb25NZXRhRGF0YQgAAAAAAhkdXJhdGlvbgAAAAAAAAAAAAAFd2lkdGgAAAAAAAAAAAAABmhlaWdodAAAAAAAAAAA
AAAMdmlkZW9jb2RlY2lkAgAESDI2MwAMYXVkaW9jb2RlY2lkAgAEbm1vcwAOdmlkZW9mcmFtZXJhdGUA//gAAAAAAAAA
D2F1ZGlvc2FtcGxlcmF0ZQBAs4gAAAAAAAANYXVkaW9jaGFubmVscwA/8[AAAA
AAAAAAJdHJhY2tpbmZvCgAAAAMDAAZsZW5ndGgAAAAAAAAAAAAACXRpbWVzY2FsZQBAj0AAAAAAAAIbGFuZ3VhZ2UCA
ANlbmcAAAkDAAZsZW5ndGgAAAAAAAAAAAAACXRpbWVzY2FsZQBAj0AAAAAAAAIbGFuZ3VhZ2UCAANlbmcAAAkDAAZsZ
W5ndGgAAAAAAAAAAAAACXRpbWVzY2FsZQBAj0AAAAAAAAIbGFuZ3VhZ2UCAANlbmcAAAkAAAk]=
        </metadata>
    </media>
    <media
        streamId="test2"
        bitrate="2000"
        url="../../../../../streams/Apps/live/streams/definst/test2/test2"
        bootstrapInfoId="bootstrap7537">
        <metadata>
AgAKb25NZXRhRGF0YQgAAAAAAhkdXJhdGlvbgAAAAAAAAAAAAAFd2lkdGgAAAAAAAAAAAAABmhlaWdodAAAAAAAAAAA
AAAMdmlkZW9jb2RlY2lkAgAESDI2MwAMYXVkaW9jb2RlY2lkAgAEbm1vcwAOdmlkZW9mcmFtZXJhdGUA//gAAAAAAAAA
D2F1ZGlvc2FtcGxlcmF0ZQBAs4gAAAAAAAANYXVkaW9jaGFubmVscwA/8[AAAAAAAAAAJdHJhY2tpbmZvCgAAAAMDAAZ
sZW5ndGgAAAAAAAAAAAAACXRpbWVzY2FsZQBAj0AAAAAAAAIbGFuZ3VhZ2UCAANlbmcAAAkDAAZsZW5ndGgAAAAAAAA
AAAAACXRpbWVzY2FsZQBAj0AAAAAAAAIbGFuZ3VhZ2UCAANlbmcAAAkDAAZsZW5ndGgAAAAAAAAAAAAACXRpbWVzY2F
sZQBAj0AAAAAAAAIbGFuZ3VhZ2UCAANlbmcAAAkAAAk]=
        </metadata>
    </media>
    <media
        streamId="test3"
        bitrate="3000"
        url="../../../../../streams/Apps/live/streams/definst/test3/test3"
        bootstrapInfoId="bootstrap7538">
        <metadata>
AgAKb25NZXRhRGF0YQgAAAAAAhkdXJhdGlvbgAAAAAAAAAAAAAFd2lkdGgAAAAAAAAAAAAABmhlaWdodAAAAAAAAAAA
```

```
AAAMdmlkZW9jb2RlY2lkAgAESDI2MwAMYXVkaW9jb2RlY2lkAgAEbm1vcwAOdmlkZW9mcmFtZXJhdGUA//gAAAAAAAAA
D2F1ZGlvc2FtcGxlcmF0ZQBAs4gAAAAAAAANYXVkaW9jaGFubmVscwA/8[AAAAAAAAAAJdHJhY2tpbmZvCgAAAAMDAAZ
sZW5ndGgAAAAAAAAAAAAACXRpbWVzY2FsZQBAj0AAAAAAAAAIbGFuZ3VhZ2UCAANlbmcAAAkDAAZsZW5ndGgAAAAAAAA
AAAAACXRpbWVzY2FsZQBAj0AAAAAAAAIbGFuZ3VhZ2UCAANlbmcAAAkDAAZsZW5ndGgAAAAAAAAAAAAACXRpbWVzY2F
sZQBAj0AAAAAAAAIbGFuZ3VhZ2UCAANlbmcAAAkAAAk]=
        </metadata>
    </media>
    <bootstrapInfo
        profile="named"
        url="../../../../../streams/Apps/live/streams/definst/test1/test1.bootstrap"
        id="bootstrap7536" />
    <bootstrapInfo
        profile="named"
        url="../../../../../streams/Apps/live/streams/definst/test2/test2.bootstrap"
        id="bootstrap7537" />
    <bootstrapInfo
        profile="named"
        url="../../../../../streams/Apps/live/streams/definst/test3/test3.bootstrap"
        id="bootstrap7538" />
</manifest
```

### More Help topics

"Configuring the server for live packaging" on page 29

"Tutorial: Live HTTP Dynamic Streaming" on page 9

## Manifest files

To stream media over HTTP, the OSMF Player requests a Flash Media Manifest file (.f4m). The manifest file contains information about a Flash media asset. The manifest file contains information about each stream in a multiple stream event. This information includes the location of the media, DRM additional header data, media bootstrap information, adaptive streaming bitrates, and so on.

In the live use cases, the HTTP Origin Module creates the manifest file as the live stream is packaged and written to disk. To create the manifest file, the HTTP Origin Module uses metadata from the Event.xml file (if it exists), the Manifest.xml file, and data from the .f4f, .f4x, .meta, .bootstrap, and .drmmeta files.

The manifest file is not written to disk. However, if the state of the recording is intact, a player can request the manifest file. When the player requests the manifest file, the HTTP Origin Module generates it.

For the state of the recording to be intact, the files must not be moved from their original locations.

***Note:*** *For information about configuring the HTTP Origin Module for live streaming, see "Configure live streaming" on page 16.*

To generate the manifest file, the HTTP Origin Module follows these steps:

1. Combine the request URL and the `HttpStreamingLiveEventPath` directive to locate the live event.
2. Retrieve metadata about the event from the Event.xml file and the Manifest.xml file.
3. Scan the event directory for stream record files (.stream).
4. Retrieve the path to the corresponding content from each .stream file. Each .stream file becomes a `<media>` element in the manifest file.
5. Retrieve metadata from the .meta file.
6. Create links to the bootstrap information and DRM additional header data (if the content is protected).

**7** Return the generated manifest document (.f4m).

***Note:*** *The HTTP Origin Module uses the* `HttpStreamingContentPath` *to map the physical path of the content (.f4f) into an absolute URL path for the video player. OSMF Player has abstracted this logic. To use OSMF player, pass only an .f4m URL.*

## Sample manifest file for a live, multi-bitrate, protected event

The following is a sample manifest file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<manifest xmlns="http://ns.adobe.com/f4m/1.0">
    <id>live_mbr_event</id>
    <streamType>live</streamType>
    <duration>0</duration>
    <bootstrapInfo
        profile="named"
        url="/live/streams/myStream.bootstrap"
        id="bootstrap2267" />
    <drmAdditionalHeader
        url="/live/streams/myStream.drmmeta"
        drmContentId="live_mbr_event"
        id="drmMetadata9996" />
    <media>
        url="/live/streams/myStream"
        bitrate="408"
        bootstrapInfoId="bootstrap2267"
        drmAdditionalHeaderId="drmMetadata9996">
        <metadata>
AgAKb25NZXRhRGF0YQgAAAAAAhkdXJhdGlvbgBARo9cKPXCjwAFd2lkdGgAQJQAAAAAAAAABmhlaWdodABAhoAAAAAA
AAAMdmlkZW9jb2RlY2lkAgAEYXZjMQAMYXVkaW9jb2RlY2lkAgAEbXA0YQAKYXZjcHJvZmlsZQBAWQAAAAAAAAIYXZj
bGV2ZWwAQEAAAAAAAAAABmFhY2FvdAAAAAAAAAAAAAAAOdmlkZW9mcmFtZXJhdGUAQD34U+JVaygAD2F1ZGlvc2FtcGxl
cmF0ZQBA53AAAAAAAAANYXVkaW9jaGFubmVscwBAAAAAAAAAAAAJdHJhY2tpbmZvCgAAAAIDAAZsZW5ndGgAQTSinwAA
AAAACXRpbWVzY2FsZQBA3UwAAAAAAAAIbGFuZ3VhZ2UCAANlbmcAAAkDAAZsZW5ndGgAQUCGAAAAAAAACXRpbWVzY2Fs
ZQBA53AAAAAAAAAIbGFuZ3VhZ2UCAANlbmcAAAkAAAk=
        </metadata>
    </media>
    <bootstrapInfo
        profile="named"
        url="/live/streams/myStream1.bootstrap"
        id="bootstrap7975" />
    <media>
        url="/live/streams/myStream1"
        bitrate="1108"
        bootstrapInfoId="bootstrap7975"
        drmAdditionalHeaderId="drmMetadata9996">
        <metadata>
AgAKb25NZXRhRGF0YQgAAAAAAhkdXJhdGlvbgBARo9cKPXCjwAFd2lkdGgAQJQAAAAAAAAABmhlaWdodABAhoAAAAAA
AAAMdmlkZW9jb2RlY2lkAgAEYXZjMQAMYXVkaW9jb2RlY2lkAgAEbXA0YQAKYXZjcHJvZmlsZQBAWQAAAAAAAAIYXZj
bGV2ZWwAQEAAAAAAAAAABmFhY2FvdAAAAAAAAAAAAAAAOdmlkZW9mcmFtZXJhdGUAQD34U+JVaygAD2F1ZGlvc2FtcGxl
cmF0ZQBA53AAAAAAAAANYXVkaW9jaGFubmVscwBAAAAAAAAAAAAJdHJhY2tpbmZvCgAAAAIDAAZsZW5ndGgAQTSinwAA
AAAACXRpbWVzY2FsZQBA3UwAAAAAAAAIbGFuZ3VhZ2UCAANlbmcAAAkDAAZsZW5ndGgAQUCGAAAAAAAACXRpbWVzY2Fs
ZQBA53AAAAAAAAAIbGFuZ3VhZ2UCAANlbmcAAAkAAAk=
        </metadata>
    </media>
</manifest>
```

The player uses the manifest file to request a content fragment. In this example, there are two streams associated with the live event, “myStream” and “myStream1”. The `<media>` elements provide the absolute URL path to the content location with the prefix “/live/streams”.

If .bootstrap and .drmmeta are found in the same location as the .f4f file, `<bootstrap>` and `<drmAdditionalHeader>` elements are included in the manifest file. The .drmmeta file can be shared across multiple streams, therefore there is only one `<drmAdditionalHeader>` element in the sample manifest file. Both myStream and myStream1 refer to the same .drmmeta file through the `drmAdditionalHeaderId` attribute.

The `metadata` element contains the metadata for one piece of media in Base64 encoding. The metadata is the same information dispatched in the ActionScript `NetStream.onMetaData()` event.

For more information, see the F4M File Format Specification at adobe.com.

## Stream record files

A stream record file (.stream) is an XML document that contains the physical location of the stream. The server creates the stream record file when an incoming stream is associated with a live event. The server creates the file with an encoded name in the following location:

applications/*appname*/events/*appinstancename*/*eventname*/*MTg1ODAyNjgwNg*=.stream

The HTTP Origin Module reads the stream record file to locate the content for the stream. The following is the format of the .stream file:

```
<?xml version="1.0" encoding="UTF-8"?>
<stream xmlns="http://www.adobe.com/liveevent/1.0">
    <type>
        f4f
    </type>
    <name>
        livestream
    </name>
    <path>
        C:\Program Files\Adobe\Flash Media Server
3.8\applications\myapp\streams\_definst_\livestream
    </path>
</stream>
```