

ADOBE® FLASH® MEDIA INTERACTIVE SERVER PLUG-IN DEVELOPER GUIDE

© 2007 Adobe Systems Incorporated. All rights reserved.

Adobe® Flash® Media Interactive Server Plug-in Developer Guide

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

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

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

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

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

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are the property of their respective owners.

Portions include software under the following terms:

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

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

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

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

Contents

Developing Plug-ins

Working with plug-ins 1

General development tasks 3

Developing an Access plug-in 4

Developing an Authorization plug-in 6

Developing a File plug-in 15

Developing Plug-ins

Adobe® Flash® Media Interactive Server 3 and Adobe® Flash® Media Development Server 3 provide a plug-in architecture written in C++ that lets you extend the functionality of the server. Use the Access, Authorization, and File plug-ins to build unique Flash Media Interactive Server and Flash Media Development Server deployments with expanded access, authorization, and file management solutions.

For example, you can use the plug-ins to accept, reject, or redirect clients before they reach your application-level code (Access plug-in), control access to streams and server events (Authorization plug-in), and create a file I/O mechanism (File plug-in).

For detailed information about each plug-in's API, see *Adobe Flash Media Interactive Server Plug-in API Reference*.

Working with plug-ins

Workflow for developing and deploying a plug-in

1. Modify the sample plug-in to meet your organization's needs or write your own plug-in.

See [Sample files](#).

2. Build and compile the plug-in for your platform.

Use Microsoft Visual Studio .NET 2003 or Microsoft Visual C++ 2005 to compile plug-ins in Windows. Use GNU Compiler Collection 3.4.x to compile plug-ins in Linux.

Plug-ins are implemented as shared library (DLL) files in Windows systems and as shared object (SO) files in Linux systems. The filenames are: libconnect.dll/libconnect.so, AuthModule.dll/AuthModule.so, and FileModule.dll/FileModule.so.

An Access plug-in must have the name libconnect.dll/libconnect.so. Authorization plug-ins and File plug-ins can have any name.

3. Deploy the plug-in.

See [Deploy a plug-in](#).

Sample files

The installer installs sample files for each plug-in. Use these samples to learn how to use the API or use them as starting points for your own plug-ins.

Access plug-in

The following sample files are installed to *RootInstall\samples\plugins\access*.

Filename	Description
adaptor.cpp sample.cpp adaptor.h	Sample Access plug-in C++ files and header file.
AccessModuleSample.sln AccessModuleSample.vcproj	Work files used with Microsoft Visual C++ for building an Access plug-in in a Windows environment. You can modify the files' values to reflect the practices in your environment.
StdAfx.h StdAfx.cpp	Sample includes file declaration files.

Authorization plug-in

The following sample files are installed to *RootInstall\samples\plugins\auth*.

Filename	Description
AuthModule.cpp	Sample Authorization plug-in C++ file.
AuthModule.sln AuthModule.vcproj	Work files used with Microsoft Visual C++ for building an Authorization plug-in in a Windows environment. You can modify the files' values to reflect the practices in your environment.
StdAfx.h StdAfx.cpp	Sample includes file declaration files.

File plug-in

The following sample files are installed to *RootInstall\samples\plugins\file*.

Filename	Description
SimpleFileAdaptor.cpp SimpleFileAdaptor.h	Sample File plug-in C++ file and header file.
FileModule.h	A header file that defines create and destroy functions for the plug-in.
FileUtil.cpp	Utility functions for working with files and directories.
FileModule.sln FileModule.vcproj	Work files used with Microsoft Visual C++ for building a File plug-in in a Windows environment. You can modify the files' values to reflect the practices in your environment.
StdAfx.h StdAfx.cpp	Sample includes file declaration files.

Header files

The following header files define the plug-in API and are installed to *RootInstall\samples\plugins\include*: FmsAdaptor.h, FmsAuthActions.h, FmsAuthAdaptor.h, FmsAuthEvents.h, FmsFileAdaptor.h, FmsMedia.h, and IFCAccessAdaptor.h.

Deploy a plug-in

Once deployed, a plug-in plugs into a server process. The process loads the plug-in, and the plug-in is unloaded when the process is unloaded.

- 1 Do one of the following to stop the server:

- In Windows, choose Start > Control Panel > Administrative Tools > Services. Select Flash Media Server (FMS) from the Services list and click Stop.
 - In Linux, open a shell window and go to the directory where the server is installed: `cd /opt/adobe/fms`. Enter the following: `./server stop`.
- 2 Copy the compiled plug-in DLL or SO files to one of the following folders:
- `RootInstall/modules/access`
 - `RootInstall/modules/auth`
 - `RootInstall/modules/fileio`
- Note:** Do not change the folder names. If you're deploying multiple Authorization plug-ins, copy all the plug-ins into the `/auth` folder.
- 3 Do one of the following to start the server:
- In Windows, choose Start > Control Panel > Administrative Tools > Services. Select Flash Media Server (FMS) from the Services list and click Start.
 - In Linux, open a shell window and go to the directory where the server is installed: `cd /opt/adobe/fms`. Enter the following: `./server start`.

General development tasks

Sending data from a plug-in to a log file

Call the `log()` function of the `IFmsServerContext` class in a plug-in to send your custom messages to the log files. Plug-in log files are located in the `RootInstall/logs` directory and are named `fileio.NN.log` and `authMessage.NN.log`, by default.

Note: The Access plug-in does not write any log files. There are no interfaces provided by the Access plug-in for logging.

The `log()` function has an argument that specifies whether the log message should also be logged to the system log (Windows Event Viewer or Linux syslog). The default value is false. Excessive logging to the system log can cause performance problems.

Edit the `Logging` section of the `Server.xml` configuration file (located in `RootInstall/conf`) to enable or disable each log file. Edit the `Logger.xml` file (located in `RootInstall/conf`) to add or change the configuration information, including the location and name of the log files. For detailed information, see the comments in the XML files.

Note: In addition to client connections, the Authorization plug-in logs the connections made when a server-side `Stream.play()` method is called. These connections are distinguished by the IP address `127.0.0.1` in the `access.log` files and in the `authEvent.log` files.

Retrieving data from a configuration file

Call the `getConfig()` function of the `IFmsServerContext` class in an Authorization plug-in or a File plug-in to retrieve custom data stored in the `Server.xml` configuration file. You can analyze this data in the plug-in to activate different functionality in the plug-in, depending on the server configuration.

The Authorization plug-in can retrieve data stored in the `Auth` XML element; the File plug-in can retrieve data stored in the `ASyncIo` XML element. You must store data in one element; multiple elements and nested elements are not supported. To add custom data, open the `Server.xml` file (located in the `RootInstall\conf` folder) in a text editor and add data to the appropriate XML element. Verify the XML and restart the server.

The following example adds data to the `Auth` element in the `Server.xml` file for the Authorization plug-in:

```
<Server>
  <Auth>exampleConfiguration</Auth>
  ...
</Server>
```

The following example adds data to the `ASyncIo` element in the `Server.xml` file for the Authorization plug-in:

```
<Server>
  <ASyncIo>exampleConfiguration</ASyncIo>
  ...
</Server>
```

Note: If the `Auth` or `ASyncIo` elements aren't in the `Server.xml` file, you may add them.

For an example of a call to `IFmsServerContext->getConfig()`, see the `AuthModule.cpp` file.

Handling time-critical calls

Every call from Flash Media Interactive Server or Flash Media Development Server to a File or Authorization plug-in—for example, `authorize()` in the Authorization plug-in—is time-critical. A call should be processed and returned back to the server as fast as possible, because the server remains in a pending state until the call is returned. If processing the call is time consuming and requires a wait or sleep operation, the passed arguments should be preserved and passed to a thread pool that is created by the plug-in. For an example, see the `FileModule` sample.

Developing an Access plug-in

Upgrading the Access plug-in

If you used an Access plug-in (`libconnect.dll`) with Flash Media Server 2 and you want to use the latest features, you need to rebuild the plug-in with Microsoft .NET 2003 or Microsoft Visual C++ 2005. On Linux, you need to rebuild the plug-in with GNU Compiler Collection 3.4.x. In addition, there has been a small interface change.

The following features are new in Flash Media Interactive Server 3 and Flash Media Development Server 3:

- Redirect client connections:

```
virtual void redirect( const char* sUri, const char* sReason ) = 0;
```

- Access client properties:

```
static const char* FLD_AUDIO_SAMPLE_ACCESS = "audioSampleAccess";
static const char* FLD_VIDEO_SAMPLE_ACCESS = "videoSampleAccess";
static const char* FLD_AUDIO_SAMPLE_LOCK = "audioSampleAccessLock";
static const char* FLD_VIDEO_SAMPLE_LOCK = "videoSampleAccessLock";
```

Access plug-in overview

An Access plug-in adds another layer of security to the server; it intercepts connection requests and lets you examine the client and the server to determine whether requests should be accepted, rejected, or redirected before the requests reach the server's script layer. You can only use one Access plug-in.

You can code the plug-in to accept, reject, or redirect requests based on criteria like how many users are connected to the server and the amount of bandwidth being consumed.

You can query your organization's database of users and passwords to determine which connection requests should be allowed. Once the plug-in accepts the connection, you can update the database with a record of the user's access to the server.

You can set read and write access for files and folders on the server, set permissions to access audio and video bitmap data, and inspect client properties.

Rewriting connections to the server

Both the Access and Authorization plug-ins let you authorize connections to the server. The main differences between the two plug-ins (regarding authorizing connections) are where they run and at which stage of the connection process they are active.

New connections arrive at the edge process (fmsedge), and the data in the connection message is sent to the Access plug-in (which runs in the edge process). At this stage, the plug-in has information about the client such as IP address, the originating URL of the SWF file, the connection (or *target*) URI, user agent, and so on. Using this information, the Access plug-in can accept, reject, or redirect the connection, or rewrite the target URI. Rewriting the target URI forces the connection to a different vhost, application, or application instance. The plug-in can't rewrite a connection to a different adaptor because the socket-level connection to the adaptor has already been made.

You must use the Access plug-in to force incoming connections to a different vhost, application, or application instance; it is too late to rewrite the target URI by the time the Authorization plug-in is notified. Also, since the Access plug-in runs early in the connection process, it is the most efficient way to screen connections. If you want authorization to be as lightweight as possible, use the Access plug-in, even if you don't need to rewrite the connection URI.

The Authorization plug-in runs in the core process (fmscore) after the connection has been forwarded to the application. The Authorization plug-in can accept, reject, or redirect attempts by clients to connect to applications, but it cannot rewrite connections to different URIs.

Note: Redirecting a connection is different than rewriting a connection URI. Redirecting a client sends a redirection message containing a new URI back to the client and terminates the current connection. The client then attempts to connect to the new URI.

Access plug-in connection flow

Once you've installed an Access plug-in, it is initialized with a context pointer when the server starts. The context pointer and plug-in pointer provide two-way communication between the Access plug-in and the server.

When a client attempts to connect to the server, the server determines whether or not an Access plug-in exists. If the Access plug-in is available, it examines the connection request and either authorizes, rejects, or redirects the connection. If an Access plug-in is not available, connection requests proceed as usual.

Note: There can only be one Access plug-in per Flash Media Interactive Server or Flash Media Development Server installation.

Writing the code in an Access plug-in

You can either modify the code in the sample Access plug-in file provided by Adobe, or you can write your own plug-in.

A client connection request triggers the `onAccess()` callback method in the Access plug-in. Write the code that examines connection requests, modifies client properties, and accepts, rejects, or redirects the requests in the `onAccess()` callback method.

Call `getValue()` to query the following client properties: `c-referrer`, `c-user-agent`, `s-uri`, `c-ip`, and `c-proto`. Call `setValue()` to modify the following client properties: `c-referrer`, `c-user-agent`, and `s-uri`. You can also call `setValue()` to modify the `readAccess`, `writeAccess`, `readAccessLock`, `writeAccessLock`, `audioSampleAccess`, `videoSampleAccess`, `audioSampleAccessLock`, and `videoSampleAccessLock` application properties.

Call `getStats()` to query the following server statistics: `eTOTAL_CONNECTED`, `eBYTES_IN`, and `eBYTES_OUT`.

After you've queried the properties above and written your connection logic, call `accept()`, `reject()`, or `redirect()` to allow a client to connect to an application on the server or not.

Configuring folder permissions on the server

The `Access` section of the `Application.xml` configuration file (located in each virtual host directory and possibly in application directories) lets you configure folder-level permissions for the Access plug-in. If the `FolderAccess` element is set to `true`, you cannot use the `readAccess` and `writeAccess` properties to set permissions for individual files, you can only set permissions at the folder level. The default value is `false`, which lets you set permissions for individual files, as in the following:

```
<!-- Controls libconnect.dll access configurations -->
<Application>
  ...
  <Client>
    ...
    <Access>
      <FolderAccess>false</FolderAccess>
    </Access>
    ...
  ...
</Application>
```

Developing an Authorization plug-in

Authorization plug-in overview

The Authorization plug-in authorizes client access to server events and fields associated with those events. Use the Authorization plug-in to do the following:

- Authorize connections to the server

Note: The Access plug-in can also authorize connections and is more lightweight. For more information, see [Access plug-in overview](#).

- Authorize playing a stream or seeking in a stream
- Authorize publishing a stream

- Map logical URLs to physical locations

For example, if the video player plays a stream “foo”—`ns.play("myvideos/foo")`—when the request is processed by the server, this virtual name could map to `c:\apps\vidapp\myvideos\`. The Authorization plug-in lets you remap this to a different physical location; for example, `c:\myvideos\`.

- Disconnect clients from the server
- Call a method in Server-Side ActionScript

Use the Authorization plug-in to deliver content to clients according to one or more of the following criteria:

- Geographic location
- Subscription level
- Stream origin
- Time and duration of a user’s access to specific streams

You can also use the Authorization plug-in to monitor stream quality of service (QoS). The plug-in reports live stream QoS information to an external log file, which could then be read to another custom built tool.

Using multiple Authorization plug-ins

You can use a chain of plug-ins to sequentially perform actions on the incoming event. Allocate specific types of events to individual plug-ins: for example, `auth1.dll` (or `auth1.so`) could authorize playing a stream; `auth2.dll` (or `auth2.so`) could authorize publishing a stream, and so on.

Note: *Windows is not case-sensitive; Linux is case-sensitive.*

The server loads the plug-ins in alphabetical order. When the server processes client requests for events, it follows alphabetical order (Windows is not case-sensitive; Linux is case-sensitive). Each plug-in filters the incoming requests.

Writing the code in an Authorization plug-in

When the server loads an Authorization plug-in, it expects the following entry points:

```
FmsCreateAuthAdaptor() // Creates an Authorization plug-in.
FmsDestroyAuthAdaptor() // Destroys an Authorization plug-in.
```

Implement the `IFmsAuthAdaptor` interface to process events as they occur. This class includes the methods `authorize()`, `notify()`, and `getEvents()`.

Once the plug-in is created, the server calls the `getEvents()` method to find out which events the plug-in wants to process. The server calls `getEvents()` once, and the events are good for the lifetime of the plug-in.

Some events must be authorized by the plug-in before the server executes them; other events only require the plug-in to notify the server that the plug-in received notification of the event. When authorization events occurs, the server calls the `authorize()` method on the plug-in. When notification events occur, the server calls the `notify()` method on the plug-in. The server suspends operations until the plug-in calls the `onAuthorize()` or `onNotify()` method of the `IFmsAuthServerContext` class to authorize or acknowledge the event and end the notification call.

Assigning actions to events

There are two actions you can assign to an event: `IFmsDisconnectAction` and `IFmsNotifyAction`. The `IFmsDisconnectAction` disconnects a client when the event occurs and the `IFmsNotifyAction` calls a method on the Client object or the application object in a server-side script when the event occurs.

To assign an action to an event, call `addDisconnectAction()` or `addNotifyAction()` from an `IFmsAuthEvent` instance. The following code adds the `IFmsNotifyAction` instance `pAction` to the `IFmsAuthEvent` instance `m_pAev`. The action calls `someMethod()` in a server-side script and passes it the parameter `12345`.

```
FmsVariant field;
IFmsNotifyAction* pAction = m_pAev->addNotifyAction("Notified by adaptor");
pAction->setClientId(field);
field.setString("someMethod");
pAction->setMethodName(field);
field.setU16(12345);
pAction->addParam(field);
```

You can call `addParam()` on a `IFmsNotifyAction` instance to pass parameters to a method. You must define the method in a server-side script. If you pass a value to `setClientId()`, the method is called on the server-side `Client` object. If you don't pass a value to `setClientId()`, the method is called on the server-side application object. For example, the following Server-Side ActionScript code defines `someMethod()` on the `Client` object:

```
application.onConnect = function(client){
    client.someMethod = function(msg){
        trace("inside someMethod");
    }
}
```

Mapping stream paths

One thing you may want to do within an `authorize()` or `notify()` method is remap virtual stream paths to different physical locations. For example, if client 1 and client 2 both request to play the stream `foo.flv`, two `E_FILENAME_TRANSFORM` events are called, and you can remap the stream differently for each client. For example, client 1 could remap stream `foo` to `c:\yourpath1\foo.flv` and client 2 could remap stream `foo` to `c:\yourpath2\foo.flv`.

To map a logical stream to a different physical path, you must set a parameter in the `Application.xml` file, as follows:

```
<Application>
    ...
    <StreamManager>
        ...
        <QualifiedStreamsMapping enable="true" />
    
```

Note: For more information about configuration files, see “Configuring the server” in Configuration and Administration Guide.

Every time a client plays or publishes a stream, an `E_FILENAME_TRANSFORM` event occurs. The `E_FILENAME_TRANSFORM` event lets you modify the `F_STREAM_PATH` property to change the path to a stream's physical location.

Events occur in the following order:

- 1 `E_PLAY` or `E_PUBLISH` The logical stream name may be changed here.
- 2 `E_FILENAME_TRANSFORM` The resulting physical stream name may be changed here.
- 3 Playback occurs.
- 4 `E_STOP` or `E_UNPUBLISH`

Note: Other operations may cause an `E_FILE_TRANSFORM` event. For example, when a server-side script calls `Stream.length()`, the server must find a physical location for that stream, which generates an `E_FILE_TRANSFORM` event for the plug-in.

Differences between edge and origin deployments

Authorization plug-ins can be deployed on origin servers and edge servers; the functionality is different in each case.

Origin When an Authorization plug-in is installed on an origin server, the `E_PLAY` (for a live stream) or `E_LOADSEGMENT` (for recorded stream) event is called to play the clip. To block play for a recorded stream, process the `E_LOADSEGMENT` event rather than `E_PLAY`.

Edge (Core) When an Authorization plug-in is installed on an edge server, the `E_PLAY` event is called to play streams delivered from the origin. If the `CacheDir` element is enabled in the `Vhost.xml` configuration file, the stream data is stored in a cache. The cache location (*RootInstall\cache*, by default) is stored in the `F_STREAM_PATH` property. You can change the value during the `E_FILENAME_TRANSFORM` event.

Note: *Server-Side ActionScript is not executed on edge servers, so the `NOTIFY` action cannot be used and should always return a failure status.*

Accessing events and properties

The server provides an interface to an event object, `IFmsAuthEvent`, that gives the plug-in access to events. The plug-in has access to certain client, application, and stream properties during each event. Not all properties are accessible during all events and each property is either read-only or read/write. The following table is a matrix of properties and events. The heading indicates whether the event requires notification, authorization, or both.

Note: *One table would be too wide to fit on the page so this information is divided into multiple tables.*

	<code>E_APPSTART</code> Notify	<code>E_APPSTOP</code> Notify	<code>E_CONNECT</code> Notify and Authorize
<code>F_CLIENT_URI</code>	None	None	Read-only
<code>F_CLIENT_REDIRECT_URI*</code>	None	None	Read/write
<code>F_CLIENT_ID</code>	None	None	Read-only
<code>F_CLIENT_IP</code>	None	None	Read-only
<code>F_CLIENT_SECURE</code>	None	None	Read-only
<code>F_CLIENT_VHOST</code>	None	None	Read-only
<code>F_CLIENT_REFERER</code>	None	None	Read-only
<code>F_CLIENT_PAGE_URL</code>	None	None	Read-only
<code>F_CLIENT_AMF_ENCODING</code>	None	None	Read-only
<code>F_CLIENT_USER_AGENT</code>	None	None	Read-only
<code>F_CLIENT_READ_ACCESS</code>	None	None	Read/write
<code>F_CLIENT_WRITE_ACCESS</code>	None	None	Read/write
<code>F_CLIENT_READ_ACCESS_LOCK</code>	None	None	Read/write
<code>F_CLIENT_WRITE_ACCESS_LOCK</code>	None	None	Read/write
<code>F_CLIENT_AUDIO_SAMPLE_ACCESS</code>	None	None	Read/write
<code>F_CLIENT_VIDEO_SAMPLE_ACCESS</code>	None	None	Read/write
<code>F_CLIENT_AUDIO_SAMPLE_ACCESS_LOCK</code>	None	None	Read/write
<code>F_CLIENT_VIDEO_SAMPLE_ACCESS_LOCK</code>	None	None	Read/write
<code>F_CLIENT_AUDIO_CODECS</code>	None	None	Read-only

	E_APPSTART Notify	E_APPSTOP Notify	E_CONNECT Notify and Authorize
F_CLIENT_VIDEO_CODECS	None	None	Read-only
F_CLIENT_TYPE	None	None	Read-only
F_CLIENT_PROTO	None	None	Read-only
F_CLIENT_URI_STEM	None	None	Read-only
F_APP_URI	Read-only	Read-only	Read-only
F_APP_NAME	Read-only	Read-only	Read-only
F_APP_INST	Read-only	Read-only	Read-only
F_STREAM_NAME	None	None	None
F_STREAM_PATH	None	None	None
F_STREAM_TYPE	None	None	None
F_STREAM_LENGTH	None	None	None
F_STREAM_POSITION	None	None	None
F_STREAM_IGNORE	None	None	None
F_STREAM_RESET	None	None	None
F_STREAM_QUERY	None	None	None
F_STREAM_PAUSE*	None	None	None
F_STREAM_PAUSE_TIME*	None	None	None
F_STREAM_PAUSE_TOGGLE*	None	None	None
F_STREAM_SEEK_POSITION*	None	None	None
F_STREAM_PUBLISH_TYPE*	None	None	None
F_STREAM_PUBLISH_BROADCAST*	None	None	None
F_SEGMENT_START*	None	None	None
F_SEGMENT_END*	None	None	None

* Not supported in versions earlier than Flash Media Server 3

The following are additional events.

	E_DISCONNECT Notify	E_FILENAME_TRANSFORM Notify and Authorize	E_PLAY Notify and Authorize
F_CLIENT_URI	Read-only	Read-only	Read-only
F_CLIENT_REDIRECT_URI*	None	Read-only	None
F_CLIENT_ID	Read-only	Read-only	Read-only
F_CLIENT_IP	Read-only	Read-only	Read-only
F_CLIENT_SECURE	Read-only	Read-only	Read-only

	E_DISCONNECT Notify	E_FILENAME_TRANSFORM Notify and Authorize	E_PLAY Notify and Authorize
F_CLIENT_VHOST	Read-only	Read-only	Read-only
F_CLIENT_REFERRER	Read-only	Read-only	Read-only
F_CLIENT_PAGE_URL	Read-only	Read-only	Read-only
F_CLIENT_AMF_ENCODING	Read-only	Read-only	Read-only
F_CLIENT_USER_AGENT	Read-only	Read-only	Read-only
F_CLIENT_READ_ACCESS	Read-only	Read-only	Read-only
F_CLIENT_WRITE_ACCESS	Read-only	Read-only	Read-only
F_CLIENT_READ_ACCESS_LOCK	Read-only	Read-only	Read-only
F_CLIENT_WRITE_ACCESS_LOCK	Read-only	Read-only	Read-only
F_CLIENT_AUDIO_SAMPLE_ACCESS	Read-only	Read-only	Read-only
F_CLIENT_VIDEO_SAMPLE_ACCESS	Read-only	Read-only	Read-only
F_CLIENT_AUDIO_SAMPLE_ACCESS_LOCK	Read-only	Read-only	Read-only
F_CLIENT_VIDEO_SAMPLE_ACCESS_LOCK	Read-only	Read-only	Read-only
F_CLIENT_AUDIO_CODECS	Read-only	Read-only	Read-only
F_CLIENT_VIDEO_CODECS	Read-only	Read-only	Read-only
F_CLIENT_TYPE	Read-only	Read-only	Read-only
F_CLIENT_PROTO	Read-only	Read-only	Read-only
F_CLIENT_URI_STEM	Read-only	Read-only	Read-only
F_APP_URI	Read-only	Read-only	Read-only
F_APP_NAME	Read-only	Read-only	Read-only
F_APP_INST	Read-only	Read-only	Read-only
F_STREAM_NAME	None	Read-only	Read/write
F_STREAM_PATH	None	Read/write	Read/write
F_STREAM_TYPE	None	Read/write	Read/write
F_STREAM_LENGTH	None	Read-only	Read/write
F_STREAM_POSITION	None	Read-only	Read/write
F_STREAM_IGNORE	None	Read-only	Read/write
F_STREAM_RESET	None	Read-only	Read/write
F_STREAM_QUERY	None	Read-only	Read/write
F_STREAM_PAUSE*	None	None	None
F_STREAM_PAUSE_TIME*	None	None	None
F_STREAM_PAUSE_TOGGLE*	None	None	None
F_STREAM_SEEK_POSITION*	None	None	None

	E_DISCONNECT Notify	E_FILENAME_TRANSFORM Notify and Authorize	E_PLAY Notify and Authorize
F_STREAM_PUBLISH_TYPE*	None	None	None
F_STREAM_PUBLISH_BROADCAST*	None	None	None
F_SEGMENT_START*	None	None	None
F_SEGMENT_END*	None	None	None

* Not supported in versions earlier than Flash Media Server 3

The following table contains additional events.

	E_PUBLISH Notify and Authorize	E_STOP Notify	E_UNPUBLISH Notify
F_CLIENT_URI	Read-only	Read-only	Read-only
F_CLIENT_REDIRECT_URI*	None	None	None
F_CLIENT_ID	Read-only	Read-only	Read-only
F_CLIENT_IP	Read-only	Read-only	Read-only
F_CLIENT_SECURE	Read-only	Read-only	Read-only
F_CLIENT_VHOST	Read-only	Read-only	Read-only
F_CLIENT_REFERER	Read-only	Read-only	Read-only
F_CLIENT_PAGE_URL	Read-only	Read-only	Read-only
F_CLIENT_AMF_ENCODING	Read-only	Read-only	Read-only
F_CLIENT_USER_AGENT	Read-only	Read-only	Read-only
F_CLIENT_READ_ACCESS	Read-only	Read-only	Read-only
F_CLIENT_WRITE_ACCESS	Read-only	Read-only	Read-only
F_CLIENT_READ_ACCESS_LOCK	Read-only	Read-only	Read-only
F_CLIENT_WRITE_ACCESS_LOCK	Read-only	Read-only	Read-only
F_CLIENT_AUDIO_SAMPLE_ACCESS	Read-only	Read-only	Read-only
F_CLIENT_VIDEO_SAMPLE_ACCESS	Read-only	Read-only	Read-only
F_CLIENT_AUDIO_SAMPLE_ACCESS_LOCK	Read-only	Read-only	Read-only
F_CLIENT_VIDEO_SAMPLE_ACCESS_LOCK	Read-only	Read-only	Read-only
F_CLIENT_AUDIO_CODECS	Read-only	Read-only	Read-only
F_CLIENT_VIDEO_CODECS	Read-only	Read-only	Read-only
F_CLIENT_TYPE	Read-only	Read-only	Read-only
F_CLIENT_PROTO	Read-only	Read-only	Read-only
F_CLIENT_URI_STEM	Read-only	Read-only	Read-only
F_APP_URI	Read-only	Read-only	Read-only

	E_PUBLISH Notify and Authorize	E_STOP Notify	E_UNPUBLISH Notify
F_APP_NAME	Read-only	Read-only	Read-only
F_APP_INST	Read-only	Read-only	Read-only
F_STREAM_NAME	Read/write	Read-only	Read-only
F_STREAM_PATH	Read/write	Read-only	Read-only
F_STREAM_TYPE	Read/write	Read-only	Read-only
F_STREAM_LENGTH	Read/write	Read-only	Read-only
F_STREAM_POSITION	Read-only	Read-only	Read-only
F_STREAM_IGNORE	Read-only	Read-only	Read-only
F_STREAM_RESET	Read-only	Read-only	Read-only
F_STREAM_QUERY	Read/write	Read-only	Read-only
F_STREAM_PAUSE*	None	None	None
F_STREAM_PAUSE_TIME*	None	None	None
F_STREAM_PAUSE_TOGGLE*	None	None	None
F_STREAM_SEEK_POSITION*	None	None	None
F_STREAM_PUBLISH_TYPE*	Read/write	None	Read-only
F_STREAM_PUBLISH_BROADCAST*	Read/write	None	Read-only
F_SEGMENT_START*	None	None	None
F_SEGMENT_END*	None	None	None

* Not supported in versions earlier than Flash Media Server 3

The following table contains additional events.

	E_SEEK Notify and Authorize	E_PAUSE Notify and Authorize	E_LOADSEGMENT Notify and Authorize
F_CLIENT_URI	Read-only	Read-only	Read-only
F_CLIENT_REDIRECT_URI*	None	None	None
F_CLIENT_ID	Read-only	Read-only	Read-only
F_CLIENT_IP	Read-only	Read-only	Read-only
F_CLIENT_SECURE	Read-only	Read-only	Read-only
F_CLIENT_VHOST	Read-only	Read-only	Read-only
F_CLIENT_REFERER	Read-only	Read-only	Read-only
F_CLIENT_PAGE_URL	Read-only	Read-only	Read-only
F_CLIENT_AMF_ENCODING	Read-only	Read-only	Read-only
F_CLIENT_USER_AGENT	Read-only	Read-only	Read-only

	E_SEEK Notify and Authorize	E_PAUSE Notify and Authorize	E_LOADSEGMENT Notify and Authorize
F_CLIENT_READ_ACCESS	Read-only	Read-only	Read-only
F_CLIENT_WRITE_ACCESS	Read-only	Read-only	Read-only
F_CLIENT_READ_ACCESS_LOCK	Read-only	Read-only	Read-only
F_CLIENT_WRITE_ACCESS_LOCK	Read-only	Read-only	Read-only
F_CLIENT_AUDIO_SAMPLE_ACCESS	Read-only	Read-only	Read-only
F_CLIENT_VIDEO_SAMPLE_ACCESS	Read-only	Read-only	Read-only
F_CLIENT_AUDIO_SAMPLE_ACCESS_LOCK	Read-only	Read-only	Read-only
F_CLIENT_VIDEO_SAMPLE_ACCESS_LOCK	Read-only	Read-only	Read-only
F_CLIENT_AUDIO_CODECS	Read-only	Read-only	Read-only
F_CLIENT_VIDEO_CODECS	Read-only	Read-only	Read-only
F_CLIENT_TYPE	Read-only	Read-only	Read-only
F_CLIENT_PROTO	Read-only	Read-only	Read-only
F_CLIENT_URI_STEM	Read-only	Read-only	Read-only
F_APP_URI	Read-only	Read-only	Read-only.
F_APP_NAME	Read-only	Read-only	Read-only
F_APP_INST	Read-only	Read-only	Read-only
F_STREAM_NAME	Read-only	Read-only	Read-only
F_STREAM_PATH	Read-only	Read-only	Read-only
F_STREAM_TYPE	Read-only	Read-only	Read-only
F_STREAM_LENGTH	Read/write	Read-only	Read-only
F_STREAM_POSITION	Read-only	Read-only	Read-only
F_STREAM_IGNORE	Read-only	Read-only	Read-only
F_STREAM_RESET	Read-only	Read-only	Read-only
F_STREAM_QUERY	Read-only	Read-only	Read-only
F_STREAM_PAUSE*	None	Read-only	None
F_STREAM_PAUSE_TIME*	None	Read-only	None
F_STREAM_PAUSE_TOGGLE*	None	Read-only	None
F_STREAM_SEEK_POSITION*	Read/write	None	None
F_STREAM_PUBLISH_TYPE*	None	None	None
F_STREAM_PUBLISH_BROADCAST*	None	None	None
F_SEGMENT_START*	None	None	Read-only
F_SEGMENT_END*	None	None	Read-only

* Not supported before Flash Media Server 3

Developing a File plug-in

File plug-in overview

The File plug-in gives you complete control over where and how the server reads content from the file system. The plug-in provides an interface between the operating system's file I/O mechanism and the server. You can configure or modify the sample file to create an alternative to the default operating-system-based file system I/O.

Previous versions of the server supported synchronous access to the file system. Each request for a read operation on a file had to wait for the previous requests in the queue to be completed. The File plug-in supports asynchronous access, making it easier to implement network-based file I/O.

You can code the File plug-in to do the following:

- Grab files from a remote location over HTTP and serve them to clients through the core server process to off-load content management duties.
- Remap files to a different physical location.

Note: *The File plug-in only works with stream files. For a list of supported file formats, see “Supported file formats” in Technical Overview.*

If a custom File plug-in is not present or is inactive, the server uses the standard operating system file system for backwards compatibility. You can only use one File plug-in.

Responding to server calls

The File plug-in is asynchronous—when the server calls the plug-in, the plug-in doesn't respond immediately. The server calls the plug-in, and the plug-in calls the server back on the response interface.

When the server calls a method on the plug-in, the plug-in interface can return an error code (-1) indicating that the operation failed. If the method returns an error, the plug-in should not call the server back.

If a call to the plug-in returns successfully, the plug-in should call the server back and pass the context received from the server.

The `close()` and `remove()` calls are an exception to this rule. When the server calls the `close()` and `remove()` methods on the plug-in, the server is not interested in the response. To free resources associated with the request before the callback occurs, the server sometimes passes `NULL` as the `pCtx` pointer. If the context is `NULL`, then it is not necessary to call the server back. If the plug-in calls the server back with the `NULL` context, the server ignores it.

Index

Symbols

.NET 2003 1, 4

A

Access plug-in
 about 5
 configuring folder permissions 6
 access.dll/access.so. *See* Access plug-in
 Application.xml file 6
 auth.dll/auth.so. *See* Authorization plug-in
 Authorization plug-in
 about 6
 edge server, deploying on 8

C

client properties, accessing 6, 9
 compiling plug-ins 1
 configuration files, retrieving data from 3

D

deploying plug-ins 2

E

events, accessing server 6, 9

F

file names, plug-in 1
 File plug-in 15
 fileio.dll/fileio.so *See* File plug-in
 Flash Media Interactive Server
 starting 2
 stopping 2
 Flash Media Server 2 5
 FmsAdaptor.h file 2
 FmsAuthActions.h file 2
 FmsAuthAdaptor.h file 2
 FmsAuthEvents.h file 2
 FmsFileAdaptor.h file 2
 FmsMedia.h file 2

G

GNU Compiler Collection 1

H

header files 2
 HTTP file access 15

I

I/O 15
 IFCAccessAdaptor.h file 2
 installation folder 2

L

Logger.xml file 3
 logging 3

M

mapping stream paths 8

P

properties, accessing client 6, 9

S

sample files 1
 security 5
 Server.xml file 3
 stream paths, mapping 8

V

Visual C++ 2005 1

X

XML configuration files, retrieving data from 3