

CHAPTER 18



Advanced Flash Media Server Techniques

Now that you know the basics of Flash Media Server, it's time to move on to more advanced features, most of which are focused on improving the user experience. For example, I will detail how to deploy a bandwidth-detection routine that enables you to serve an appropriately encoded FLV video targeted at a particular connection speed, matching file bandwidth to connection speed. You will see this routine working in ActionScript, without the use of a playback component or SMIL file.

Next up is Flash Media Server's virtual keys feature, which allows you to serve a video that fits the user's Flash Player capabilities, and in particular to match the VP6 codec to Flash Player 8+ while serving users with Flash Player 6 and 7 a Spark-encoded video. All this detection routine runs server side and removes complex Flash Player version detection logic from your SWF movie.

After deploying your Flash videos, you'll probably want to know how well they're performing—and if not *you*, then certainly your boss or client will want to. Fortunately, Flash Media Server now supports W3C-compliant log files in text format, which simplifies reporting, and also features a much improved admin console so you can monitor your server's vital health issues in real time.

I will conclude the chapter by looking at some additional Flash Media Server features that enable it to be a conferencing platform, multiuser game server or notification system (anyone remember push technology?), to name but a few.

To sum up, here's a list of what I will cover in this chapter:

- Using virtual directories to organize FLV files
- Serving streams according to Flash Player version using virtual keys
- Using bandwidth detection

- Logging and reporting
- The power of SharedObjects.

Let's jump right in.

ALERT



This chapter uses Adobe Flash CS3 Professional (alternatively you may use Adobe Flash Professional 8). To follow along with the examples, which I recommend, you will need a copy of the program installed on your computer. Should you not own a copy of the program, note that Adobe offers a free 30-day trial version on its website.

Also, note that you can find the files referred to and used in this chapter, including completed skins, at www.flashvideobook.com. Consider downloading these files now, so they'll be available once I start referring to them.

Using Virtual Directories to Organize FLV Files

Virtual directories in Flash Media Server allow you to share resources like FLV files between applications and application instances. They are essentially a mapping of a keyword to a location on the server's hard drive, attached storage device or even a network storage location. This allows several different video players access to a large library of video content and is often essential for large-scale deployments.

You may recall from the last two chapters that by default a Flash Media Server application can only access resources that are stored in a specific folder, which is named according to the current application instance (which by default is called `_definst_`). If you can't remember where FLV files are stored on the server, then now is a good time to skip back to Chapter 16 and the section titled "Examining the Server's Folder Structure."

Virtual directories allow you to break out of this strict folder structure that the default set-up imposes. Now that you understand the default set-up and basic streaming techniques, you can work up to a finer tuned configuration.

Updating the Server Configuration

To make use of virtual directories, you must configure them in a file called `vhost.xml`, which is located in the server's installation directory. The path to this file (from your installation root) is `/conf/_defaultRoot/_defaultVHost/Vhost.xml` as shown in **Figure 18-1**.

Follow these steps to configure a virtual directory for your FLV videos:

1. Create a new folder `myvideos` on your primary hard drive. In my case this is the C: drive. You may create this folder in another location and/or give it a different name; note, however, that in this example I will refer to the location of `C:\myvideos` and I recommend you follow this naming convention to make the example easier to follow.

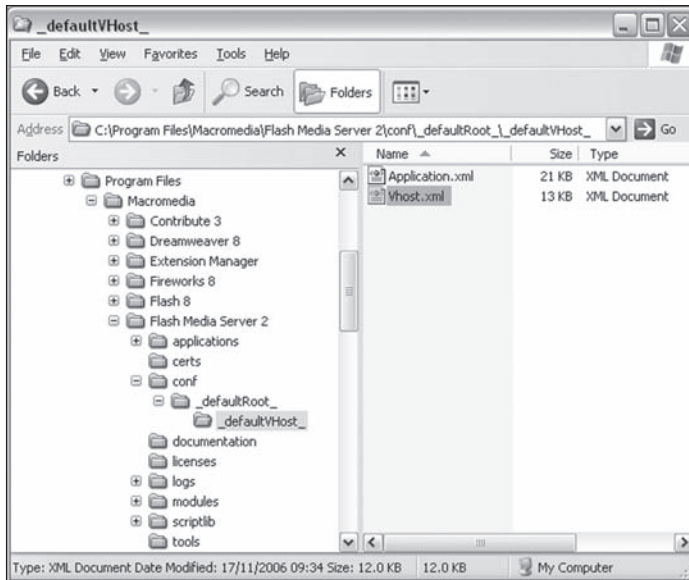


Figure 18-1
Vhost.xml shown in its default location.

2. Locate the folder videos among the sample files for this chapter. Again, you can download all example files for the book from www.flashvideobook.com.
3. Inside the videos folder, you will find a file sample_250kbps.flv. Copy this file into the newly created folder C:\myvideos, as shown in **Figure 18-2**.

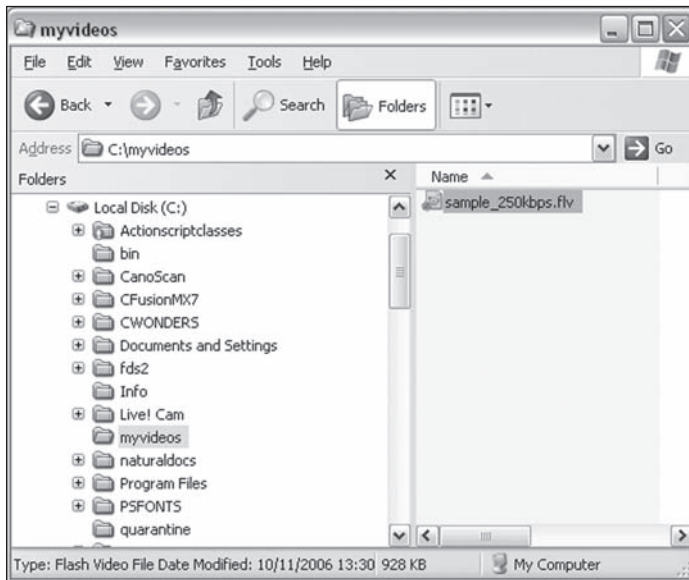


Figure 18-2
You created C:\myvideos to contain video files and copied in the file sample_250kbps.flv.

4. Open the Vhost.xml file in your favorite text editor.
5. Find the <VirtualDirectory> tag by scrolling down to around line 65. Note that any text contained between <!-- and --> is a comment and has no effect on server configuration.
6. Locate the <Streams></Streams> tag nested within the <VirtualDirectory> tag.
7. Modify the <Streams> tag so that it reads:

```
<Streams>mymapping;C:\myvideos</Streams>
```

You have now created a mapping by the name of mymapping, which resolves to the location of C:\myvideos. This means that every time a Flash application requests a stream starting with the name mymapping, the server will look in C:\myvideos for the file.

8. Restart the server by logging on to the admin console and choosing the Manage Servers option (consult Chapter 17 for details on how to access the admin console). Then click the restart button on the bottom left, as shown in **Figure 18-3**.

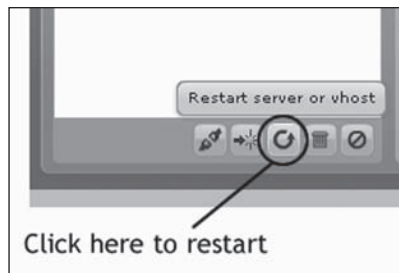


Figure 18-3

Click this button to restart the server.

The server must restart to reload the XML configuration files and for the changes to take effect. You can now test the newly created virtual directory in Flash.

9. Open the file virtual.fla in Flash. You can find it among the source files for Chapter 18 in a folder called virtual_directory.
10. Click frame 1 of the actions layer to select it.
11. Choose Window > Actions from the menu bar to show the Actions panel if it is not already showing.
12. Inside the Actions panel, scroll down to line 25 to the following line of code:

```
ns.play("mymapping/sample_250kbps");
```

The path to this video file starts with the name of the virtual directory mapping that you have just created. This means that when this application executes it will instruct the server to check in C:\myvideos for a file called sample_250kbps.flv (note that the .flv extension needs to be omitted from the FLV name when using ns.play in ActionScript).

Testing the Configuration

OK, we've got it all set up; now let's test.

1. Choose Control > Test Movie from the menu bar to test the application. Flash will compile and run the file and the video should start playing.

Should the video fail to play, verify that the RTMP address on line 35 points to a valid application on your server. Also make sure that the server is running and that you have followed the previous steps correctly.

Verifying the Mapping

To verify that the server can access files outside the default location of the applications directory (which is normally /streams/instancename), you can modify the RTMP string on line 35:

```
nc.connect("rtmp://localhost/streamvideo");
```

Currently this line of code will connect the application to the default instance `_definst_`.

2. Add an instance name by appending "myinst" to the RTMP address so that it reads `nc.connect("rtmp://localhost/streamvideo/myinst");`
3. Test the application again by choosing Control > Test Movie from the menu bar.

You will see that the same video plays, unrelated to which instance name you are connecting to. Note that the mapping is available to all other applications on the same Vhost (and in most cases this means to all applications on the entire server). As long as any requested stream name is `mymapping/sample_250kbps` then the server will find and play that file, regardless of the application's name. This is very different from the standard set-up, which ties the location in which streams are stored to a particular application.

You have now successfully created and configured a virtual directory mapping and have used it to break out of the default directory structure, enabling you to share video files among applications and application instances. In the next section you will build on this knowledge and use a feature called virtual keys to expand on the functionality of virtual directories.

Author's Tip

Mapping Network Drives

You can also use virtual directories to map to storage locations that are not on the same physical machine as Flash Media Server. A common set-up is to map locations on network drives, enabling large amounts of storage space to be added without requiring hard-drive space on the server itself.

You should note that the syntax used to map a network drive is crucial. You should avoid using external drive letters or network share names, but instead use the IP address of the device that you are mapping. The correct syntax to map a network drive is therefore: `<Streams>mymapping;\\192.168.0.5\foldername</Streams>` (you need to substitute the IP address used in this example with the actual IP address of your device). You must also make sure that the Flash Media Server has read and write permissions for the location you are mapping.

Serving Streams According to Flash Player Version Using Virtual Keys

The virtual keys feature in Flash Media Server enables on-the-fly mapping of stream directories to match the user's Flash Player version. Using virtual keys in combination with virtual directory mappings enables you to deploy a custom stream-delivery feature that will serve Spark-encoded videos to users who have not installed the required Flash Player to view VP6-encoded videos. Likewise VP6-encoded videos can be served to those users who have the appropriate Flash Player installed. The Flash Media Server manages the entire detection and mapping routine on the fly.

Server-Side Flash Player Detection

Each Flash Player (which represents a user) that connects to Flash Media Server is automatically assigned a virtual key (once configured) based on the detected Flash Player version.

The server can read the Player version based on a parameter that is automatically passed to it in the form of a FlashVar string. The specifics of this process needn't concern you; you only need to be aware of the fact that the server can "read" the Flash Player's version number.

Configuring Virtual Keys

For the purpose of this exercise I will presume that you want to detect and divide your users into two groups: those who can view video encoded with VP6, the later and higher-quality Flash video codec, and those who can't, who will be served Spark-encoded footage. This is also the most commonly used configuration.

Author's Tip

Just as a reminder: while Flash Player 6 supports streaming video playback using Spark, it does not support progressive playback. To support progressive Flash video, your users need to have a minimum of Flash Player 7 installed.

The earliest Flash Player to support VP6 playback, progressive or streaming, was Flash Player 8. You therefore need to create two virtual keys, one covering Flash Player 8 and above and another covering Flash Player 6 to 7, both of which support Spark-encoded video when streamed.

To set up virtual keys, you first need to create the appropriate directory and key mappings in the server's XML configuration. Here's the procedure:

1. Create a new folder called `virtualkeys` inside the folder `C:\myvideos`, which you created in the previous section. If you haven't followed along with the previous section, then I suggest you now refer to step 1 of the previous exercise.
2. Locate the sample files for Chapter 18, which contain a directory by the name of `virtual_keys`.

3. Open the virtual_keys directory where you will find two folders, player6_7 and player8.
4. Copy both folders including contents into C:\myvideos\virtualkeys.

Figure 18-4 shows the correct set-up. Of course, you may create these folders in a location of your choice as long as the mapping you create in the next steps reflects this location.

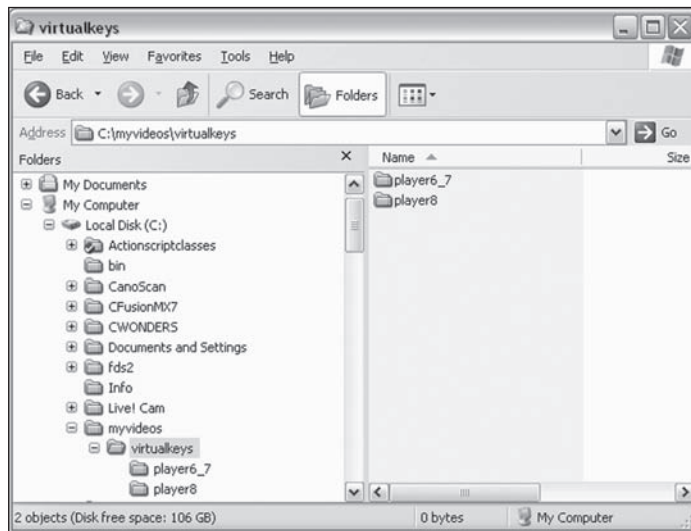


Figure 18-4
Your mappings will target these newly created directories.

You've probably already guessed that the folder player6_7 would contain videos targeted at Flash Player 6 and 7 (encoded using Spark) and the folder player8 would hold videos targeted at Flash Player 8, encoded using VP6. Currently each folder contains one video called sample_250kbps.flv.

Note that only the names are identical; the actual content is different so you can later distinguish and test which folder is serving the files. In a production environment, both directories would normally hold identical footage, the only difference being the codec used for encoding the footage.

5. Open your server's Vhost.xml file in your favorite text editor. This file can be found in Flash Media Server's default installation directory in the subfolder /conf/_defaultRoot/_defaultVHost_. **Figure 18-5** shows this location once again.
6. Find the <VirtualKeys> node at around line 42. You will now create two keys, A and B, to match the Flash Player versions you want to detect and redirect.

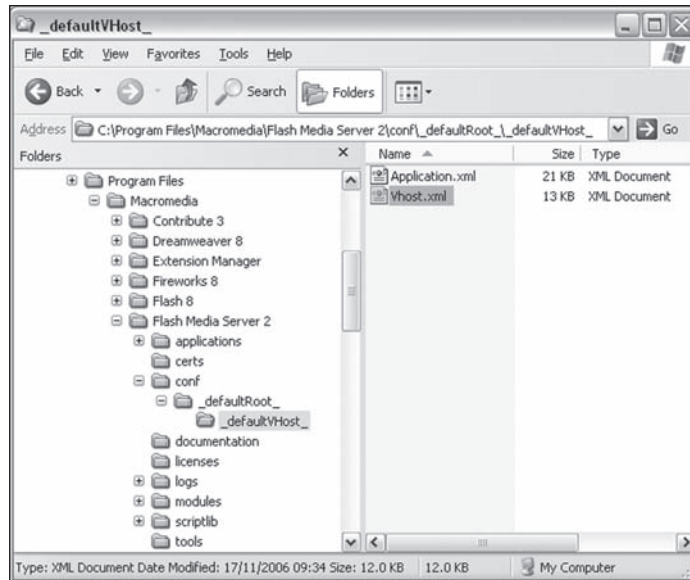


Figure 18-5
You can find Vhost.xml in this location.

Author's Tip

Assigning Virtual Keys at Runtime

Virtual keys can also be set in a server-side main.asc script. This allows you to inject more advanced logic into the assignment of virtual keys, for example setting a virtual key based on the user's detected bandwidth. For more information on this, please consult the Flash Media Server documentation.

As previously mentioned, Content Delivery Networks do not allow for a custom main.asc to be deployed and the on-the-fly assignment of virtual keys is therefore not possible when using CDN delivery.

7. Add four key tags to the VirtualKeys node so that it reads:

```
<VirtualKeys>
<Key from="WIN 6,0,0,0" to="WIN 7,9,9,9">A</Key>
<Key from="WIN 8,0,0,0" to="WIN 20,0,0,0">B</Key>
<Key from="MAC 6,0,0,0" to="MAC 7,9,9,9">A</Key>
<Key from="MAC 8,0,0,0" to="MAC 20,0,0,0">B</Key>
</VirtualKeys>
```

This creates two virtual keys, A and B, for both Windows and Macintosh platforms. Key A covers Flash Player 6 and 7 while key B covers Flash Player 8 to 20. I've used a figure of 20 to make this configuration compatible with future Flash Player versions.

8. Scroll down in Vhost.xml to around line 65 and find the <VirtualDirectory> node. You will now configure two virtual directories to map to the keys and folders you created in the previous steps.

9. Add two new <Streams> nodes to the <VirtualDirectory> node so that it reads

```
<Streams key="A">playermap;C:\myvideos\virtualkeys\player6_7</Streams>  
<Streams key="B">playermap;C:\myvideos\virtualkeys\player8</Streams>
```

Make sure to restart Flash Media Server if it has been running while you were making these configuration changes. Otherwise, start it up via the Services panel or choose Start > All Programs > Macromedia > Flash Media Server 2 > Start Flash Media Server 2.

Testing the Application

I have created a simple player that will allow you to test the newly created configuration. The application is essentially identical to the player from the previous exercise, which in turn is based on the file streamvideo_as.fla from Chapter 17.

You can find the application among the sample files for this chapter. The filename is virtual_keys.fla and it's located inside the virtual_keys folder.

1. Open the file virtual_keys.fla in Flash 8 or Flash CS3. The main differences in this file compared to the previous example on virtual directories are:
 - The mapping name you will be using now is playermap:

```
ns.play("playermap/sample_250kbps");
```

Remember that the name of the mapping is entirely up to you. For this example I chose playermap simply because I want to make it clear that this mapping is a different one from the one used in the previous section.
 - The FLA's publish settings have been changed to Flash Player 6 as this is the minimum version required to play the Spark-based video files.
 - I've made some minor code changes to make the ActionScript compatible with Flash Player 6, as this is the lowest Flash Player version that this application needs to support.

Let's remind ourselves what is going to happen when you compile this application.

- Flash Player will automatically notify the server about its version and platform.
 - Flash Media Server will notice the mapping playermap that is passed in when a stream is requested and look for (and find) a mapping pointing to two virtual key mappings, defined in Vhost.xml.
 - The server will match the Player version to either key A or B, resulting in an automatic streams directory mapping to either C:\myvideos\virtualkeys\player6_7 or C:\myvideos\virtualkeys\player8
 - The server will then serve the appropriate file.
2. Test the application in your default browser by choosing File > Publish Preview > Default (HTML) from the menu bar. The application compiles and runs in your

Author's Tip

Note that this exercise presumes that the previously created stream-video application is still present. However, any other application will also work fine for testing this file due to the fact that stream mappings work independently of the application name.

default browser. If you have Flash Player 8 or above installed in your browser, then you should now see a video of Jan talking about green-screen lighting. If your Flash Player version is 6 or 7, then you should see a short advertising snippet.

Testing with a Different Flash Player

You probably realize now that this application is tricky to test unless you uninstall and reinstall different Flash Player versions—otherwise, how can you verify that the mapping switches to the correct location? If you are using Firefox as your preferred browser and Windows as your operating system, then you can very easily overcome this by installing a great extension called Flash Switcher by

Alessandro Crugnola. It allows for easy and painless switching of Flash Player versions at the click of a button. You can download Flash Switcher at <http://tinyurl.com/y1lmet9>.

If you are not using Firefox on Windows and haven't got access to a different machine for testing purposes, then you may have to uninstall and reinstall the Flash Player. You can download archived Flash Player versions from http://www.adobe.com/go/tn_14266.

If you have Flash Switcher installed and test the application using both Player 6 and Player 9, then you will see two different videos depending on your chosen Flash Player, as shown in **Figure 18-6**.

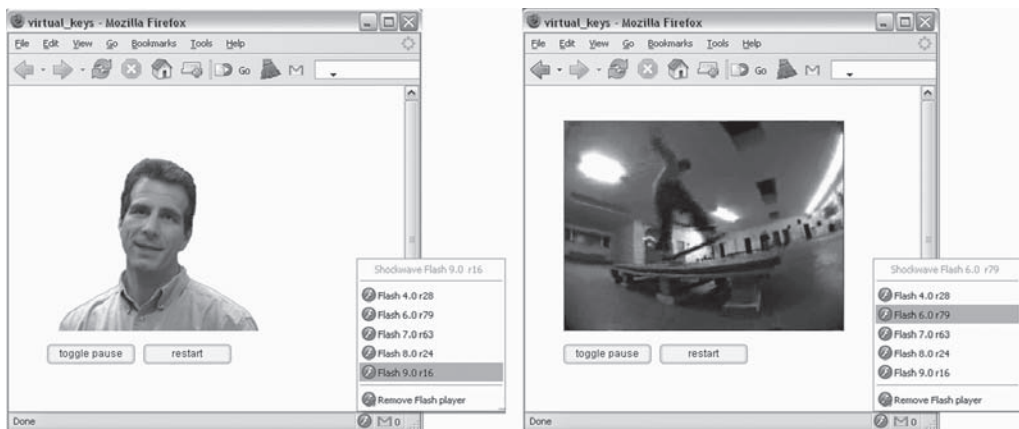


Figure 18-6

The same application plays a different file depending on the current Flash Player version.

Let me remind you once more: you normally would not serve different video but rather the same source video encoded with either Spark or VP6. The point of the exercise is to use Flash Media Server's virtual keys feature to redirect requests to the correct virtual directory on the fly.

Using Bandwidth Detection

In Chapter 17, you learned how to use SMIL files in combination with the FLV Playback component and a SMIL file to detect the user's bandwidth and to stream files to the user that are matched to that bandwidth. While this approach works well, it requires the FLV Playback component, which may not be suitable for some custom applications.

Other use cases for manual detection may include gathering bandwidth data for statistical purposes without actually serving any video content at all. And lastly, such data can also be useful for live video broadcasts—for example, to inform a dial-up user that her connection will not be sufficient to experience the event fully.

Fortunately, you can easily hook into the existing bandwidth detection scripts with a few lines of ActionScript and use the detected data however you see fit. Not only that, since the same scripts are used by most Content Delivery Networks, it is also a solution that works in a globally distributed network. All you need to do is call the right server-side functions and let the server do the rest.

Requirements for Manual Bandwidth Detection

Flash Media Server uses a server-side ActionScript file (usually called `main.asc`) to provide certain functionality to a connecting client. This `main.asc` file also contains the bandwidth detection logic, so you therefore need to deploy it like you did when you used it in combination with SMIL files and the FLV Playback component.

Creating the Application on the Server

The detection routine you will soon deploy must connect to a server-side application and you will use the `streamvideo` application from Chapter 17 for this exercise. If the `streamvideo` application still exists in your Flash Media Server's applications directory, then you're ready to go on the server side.

If you deleted the `streamvideo` application, or haven't yet set it up, then please refer to Chapter 17 and the section titled "On-Demand Streaming Using the FLV Playback Component," which details where and how to create the application.

Also make sure that the `main.asc` file is still present and, if not, then copy it back into the `streamvideo` folder. You can find the file among the sample files for Chapter 18 in a folder called `manual_bandwidth`. This `main.asc` is identical to the one used with the FLV Playback component in Chapter 17. Finally, make sure that Flash Media Server is running before moving on to the client-side application.

Using ActionScript to Detect a User's Bandwidth

The ActionScript requirements that enable your application to detect and process bandwidth checking are very simple. All you need to provide are two functions in your Flash applications, which are defined on the `NetConnection` Object.

Author's Tip

To refresh your memory on the significance of the `main.asc` file, you can skip back to Chapter 17 and read the section titled "On-Demand Streaming Using the FLV Playback Component."

When the application connects to Flash Media Server, it will pass a flag (of true or false) indicating whether or not the server should detect connection bandwidth. If the flag is set to true, the server will call a function called `onBWCheck` several times to obtain an average bandwidth figure based on those calls. The `onBWCheck` function looks like this:

```
nc.onBWCheck = function()
{
    trace("onBWCheck called");
    return;
}
```

As you can see, it is extremely simple and does nothing more than return the call to the server. The trace statement will notify you during testing that the function has been successfully invoked.

Once the bandwidth check is complete, the server will call another function called `onBWDone`. This function is passed several arguments, one of which is the detected bandwidth in Kbits per second. The complete function looks as follows:

```
nc.onBWDone = function( kbitDown, deltaDown, deltaTime, latency )
{
    // more logic here
}
```

The developer can add any further processing logic to this function. Additional parameters that are passed to `onBWDone` are `deltaDown`, `deltaTime` and `latency`. The delta figures can usually be ignored but the latency figure give some insight into how responsive a user's connection is. Latency is measured in milliseconds and the lower figures indicate a healthier Connection.

Putting It All Together

To see this code in action, you can use a file named `bwcheck fla` that you can find among the sample files for Chapter 18 in a folder called `manual_bandwidth`. Let's do this together:

1. Open the file `bwcheck fla` in Flash 8 or Flash CS3. This file contains nothing more than the ActionScript needed for bandwidth detection and a text field on Stage to display the results.
2. Click frame 1 of the Actions layer to select it.

3. Choose Window > Actions to open the Actions panel if it isn't already showing. The completed code looks as follows:

```
nc = new NetConnection();
nc.onStatus = function(info)
{
    trace(info.code);
    if (info.code == "NetConnection.Connect.Success")
    {
        startTime = getTimer();
    }
};

nc.connect("rtmp://localhost/streamvideo", true);

nc.onBWDone = function( kbitDown, deltaDown, deltaTime, latency )
{
    now = getTimer();
    endTime = now - startTime;
    trace("onBWDone: kbitDown = " + kbitDown + ", deltaDown= " + del-
    taDown + ", deltaTime = " + deltaTime + ", latency = " + latency);
    stats.htmlText = "Bandwidth check completed in " + endTime/1000 + "
    seconds<br><br>Detected speed: " + kbitDown + " Kbit/sec<br><br>";
}

nc.onBWCheck = function()
{
    trace("onBWCheck called");
    return;
}

stop();
```

Most of this code should look familiar to you by now. After creating a NetConnection and associated onStatus handler, the code will connect the application to the locally running streamvideo application.

```
nc.connect("rtmp://localhost/streamvideo", true);
```

The last parameter true is a signal to the server (and to the code in main.asc that gets invoked) to run a bandwidth detection routine.

The line `startTime = getTimer();` captures a time marker which later allows us to calculate how long it took to complete the bandwidth-detection process. This step is optional and not needed for a speed result.

What follows is the previously described code consisting of the `onBWCheck` and `onBWDone` methods, which are defined on the `NetConnection` instance `nc`.

4. Choose **Control > Test Movie** to compile and run the application.

Once the Flash Player connects to the server, the bandwidth-detection routine will automatically run and display the results in the text field on Stage, as shown in **Figure 18-7**.

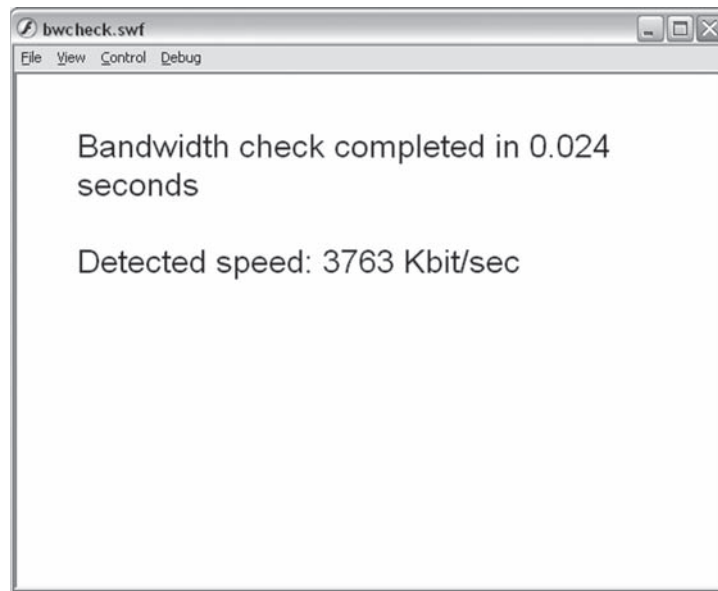


Figure 18-7

Detected bandwidth speed and time taken are displayed upon completion.

Author's Tip

Alternative detection routines can involve either a JavaScript-based approach or a timed file download. While this latter approach also works quite well, one drawback is that it involves the transfer of large chunks of data, which makes it tricky to deploy if detection is required throughout the user's session.

It can also be problematic for dial-up users because of the time it takes to complete the test. In contrast, Flash Media Server's detection routine does not involve large amounts of data and is easy to use at any stage throughout the session.

Of course, the connection speed is very high because the Flash Media Server is running locally; over the Internet, this figure would be much lower. Remember that this test measures how quickly data can flow between the server and the currently connected client at the time of the test; it does not detect the maximum speed that the client's Internet connection may support.

Your application is now ready to further process the gathered data, for example by requesting a particular video file. Other options could include a call to a remote script that might log this data to

a database for later processing. The particular implementation depends on the specifics of the application that you plan to develop.

Logging and Reporting

Log files are essentially text files containing information about system events, errors or simple process workflows. These events can include data about successful and failed connections, data flow, system state, bandwidth usage and a multitude of other data. Most servers and computer systems support some kind of logging mechanism and the data it produces is commonly used for troubleshooting and security analysis.

Before the release of Flash Media Server 2, the only format in which the server would write log files was FLV. That's right—logs were captured inside what is essentially a video file and written as data tracks, which made it tricky to process the logs efficiently.

Luckily, all this changed with the release of FMS as it introduced a text file format for all logs and, better still, produced W3C-compliant logs, which is the most common format for most web and streaming servers. This made the log files easier to process using several readily available log readers and data-mining programs.

Log Files and Storage Locations

Flash Media Server writes its logs to several different files, depending on which part of the server triggered the event being logged. Log files are, by default, written to the server's logs directory, which is a subdirectory of the installation directory (on Windows this is most commonly `C:\Program Files\Macromedia\Flash Media Server 2`).

The logged events are split into the following files:

- **Access log file (access.log)** – This file includes information about client connections (usually web users connecting to your application) as well as stream activity. It details user activity and can be used to produce reports that detail consumed bandwidth for a session, accessed resources and viewed streams, among other things.
- **Application log file (application.log)** – Application logs log data about a particular application instance. These logs are commonly used for debugging, and also capture trace messages within any server side ActionScript files (.asc). Application logs are stored in a subfolder of the logs directory named after the vhost, application name and application instance. Using the SMIL application of Chapter 17 as an example, the path to the application logs would normally be `C:\Program Files\Macromedia\Flash Media Server 2\logs_defaultVHost_smil_definst_.`

Author's Tip

W3C stands for World Wide Web Consortium, which is the main international standards organization for the World Wide Web. The W3C's mission is "To lead the World Wide Web to its full potential by developing protocols and guidelines that ensure long-term growth for the Web."

- **Diagnostic logs** – Diagnostic logs include several files, which are master.log, edge.log, core.log, admin.log and httpcache.log. Their names are fairly descriptive and these logs are most likely used to debug uncommon events that occur but are not captured by the operating system's own logs. All of the diagnostic logs are located in the logs directory inside the installation directory.

Logging Configuration

The server's logging behavior and configuration are controlled by two files, Logger.xml and fms.ini. Both files are located in the server's conf directory, which under Windows you can usually find at *C:\Program Files\Macromedia\Flesh Media Server 2\conf*.

The fms.ini file contains substitution variables used throughout the server's XML configuration files, so it is not solely used to configure logging features. All configuration entries using a dollar sign and curly bracket syntax, like `${LOGGER.LOGDIR}`, are configured in fms.ini. Fms.ini therefore offers a single point of configuration for the most commonly used options, while finer control is offered by modifying the XML files directly.

The `LOGGER.LOGDIR` directive is incidentally also the only configuration option in fms.ini that is related to logging. By default, this entry is empty, which means the logs are written to the default locations.

In contrast, the Logger.xml file offers more options than just the log-file location. It includes multiple configuration options, including specifying an external logging server to send logs to (handy for larger server clusters), maximum log file size, naming masks, events to be logged and loads of other options. For more information on this, you can consult your server documentation. You can find the online version at <http://tinyurl.com/yhew62>.

Producing Reports

It is unfortunate that Flash Media Server does not provide any reporting capabilities outside the live view enabled by the admin console. It is therefore quite challenging to produce detailed reports based on the generated log files.

Even worse, I have yet to find a tool that can digest FMS log files reliably. For example, most web-server log parsers seem to choke on FMS log files, presumably because these tools are targeted at web-server logs and not Flash Media Server.

Introducing Microsoft Log Parser

One powerful but technically challenging option for producing reports from FMS log files is a free tool from Microsoft, called Log Parser. Currently in version 2.2, Log Parser provides universal query access to text-based data such as log files, XML files and CSV files. Since FMS writes its logs in W3C compliant format, Log Parser can read and access the data contained within its log files and create reports, charts or statistics based on it. One downside of Log Parser is (of course) that it only runs on Windows.

Log Parser also supports output targets, so you can use the utility to insert the results into a database, among other things. It uses a SQL-like syntax, similar to the language used to access databases.

You can download Log Parser at <http://tinyurl.com/5uoxz> and, once installed, access it via Start > All Programs > Log Parser 2.2. I can't go through all of Log Parser's capabilities, but I will show you the most commonly used features with a sample script for each use case.

Producing Charts

Assuming that you have installed both Log Parser and Flash Media Server in their default locations, you can use the following syntax to produce a 3D column chart that lists all played videos contained within the log file access.02.log (note the use of `-i:W3C` to specify the W3C log format):

```
C:\Program Files\Log Parser 2.2>logparser "SELECT x-sname, COUNT(*) as videos into chart.gif FROM 'C:\Program Files\Macromedia\Flash Media Server 2\logs\access.02.log' GROUP BY x-sname" -i:W3C -chartType:Column3D
```

Log Parser will save the chart into its installation directory under the name of chart.gif, which you can see in **Figure 18-8**.

Author's Tip

If you are outsourcing your Flash video delivery to a third-party hosting provider or CDN then you needn't worry about reporting and logging, as your provider should handle this for you. The level of reporting varies from vendor to vendor, with more detailed reporting usually available at additional cost and basic statistics free of charge.

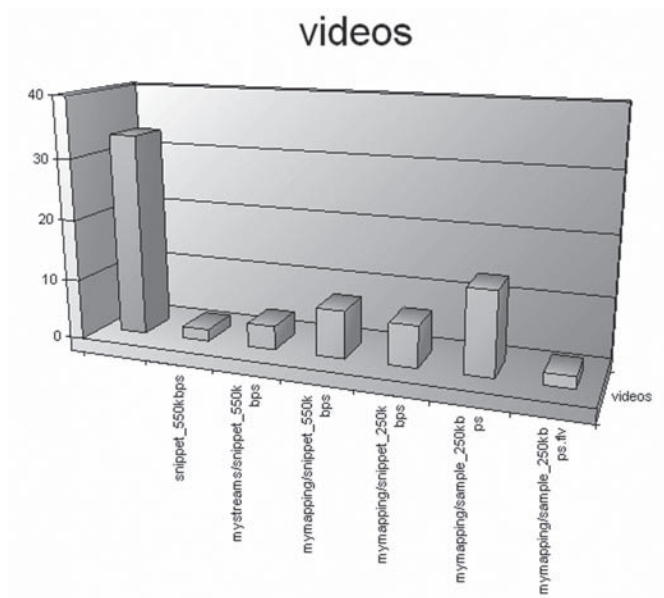


Figure 18-8
Accessed video streams are displayed in a column chart.

To process all access logs contained within the logs directory, you would simply alter the syntax to read:

```
C:\Program Files\Log Parser 2.2>logparser "SELECT x-sname, COUNT(*) as videos into chart.gif FROM 'C:\Program Files\Macromedia\Flash Media Server 2\logs\access*.log' GROUP BY x-sname" -i:W3C -chartType:Column3D
```

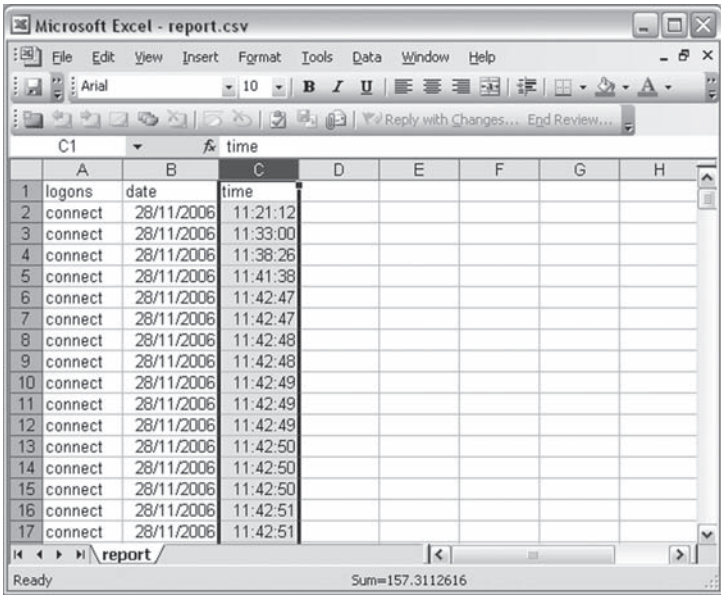
Generating Reports

Log Parser usually displays its query results right inside the output window, which isn't very handy when your goal is to produce easy-to-manage reports. Fortunately, the tool supports a variety of output formats, including CSV, XML, SQL and others.

It can also write the generated output directly to a file, as the following example shows. It produces a spreadsheet report.csv (saved into Log Parser's installation directory), which contains date and time information about each connect event across all access log files stored in FMS's default logs directory:

```
C:\Program Files\Log Parser 2.2>logparser "SELECT x-event as logons, date, time INTO report.csv FROM 'C:\Program Files\Macromedia\Flash Media Server 2\logs\access*.log' WHERE x-event='connect'" -i:W3C
```

You can see the resulting spreadsheet in **Figure 18-9**.



| | A | B | C | D | E | F | G | H |
|----|---------|------------|----------|---|---|---|---|---|
| 1 | logons | date | time | | | | | |
| 2 | connect | 28/11/2006 | 11:21:12 | | | | | |
| 3 | connect | 28/11/2006 | 11:33:00 | | | | | |
| 4 | connect | 28/11/2006 | 11:38:26 | | | | | |
| 5 | connect | 28/11/2006 | 11:41:38 | | | | | |
| 6 | connect | 28/11/2006 | 11:42:47 | | | | | |
| 7 | connect | 28/11/2006 | 11:42:47 | | | | | |
| 8 | connect | 28/11/2006 | 11:42:48 | | | | | |
| 9 | connect | 28/11/2006 | 11:42:48 | | | | | |
| 10 | connect | 28/11/2006 | 11:42:49 | | | | | |
| 11 | connect | 28/11/2006 | 11:42:49 | | | | | |
| 12 | connect | 28/11/2006 | 11:42:49 | | | | | |
| 13 | connect | 28/11/2006 | 11:42:50 | | | | | |
| 14 | connect | 28/11/2006 | 11:42:50 | | | | | |
| 15 | connect | 28/11/2006 | 11:42:50 | | | | | |
| 16 | connect | 28/11/2006 | 11:42:51 | | | | | |
| 17 | connect | 28/11/2006 | 11:42:51 | | | | | |

Figure 18-9

This spreadsheet lists connect events and their date and time.

While these are very basic examples, they nicely demonstrate Log Parser's capabilities. It wouldn't be too difficult to build upon this or other examples to gather any type of information required, format it, filter it and output it to a specific file format and location.

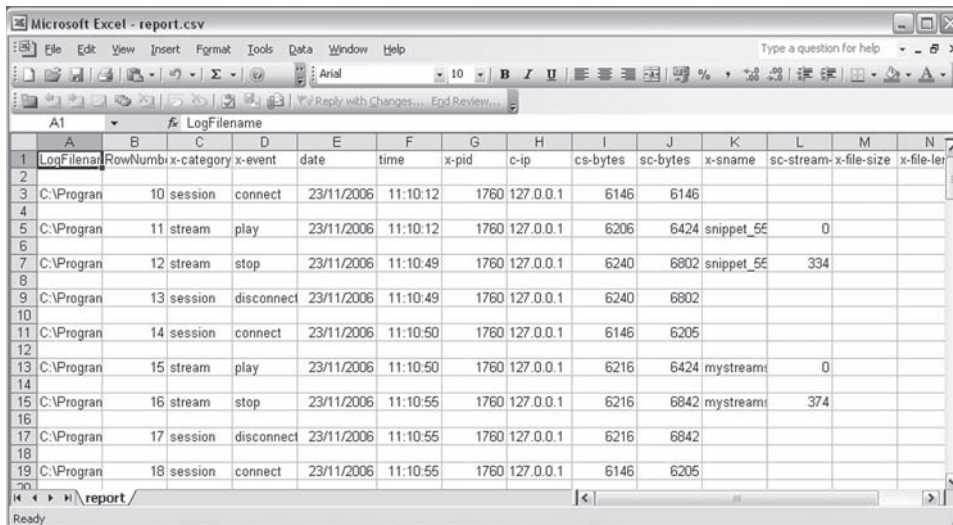
Searching for Data

The power of SQL is ideally suited to search for specific pieces of information across an entire directory, or individual files if required. The following example shows how to search for all events generated by a specific IP address, in this case 127.0.0.1, which is the localhost IP that I used during testing.

The query is run against a single access log file named access.03.log but you could also query an entire directory if needed. Again, Log Parser saves the results to report.csv as a spreadsheet.

```
C:\Program Files\Log Parser 2.2>logparser "SELECT * INTO report.csv FROM 'C:\Program Files\Macromedia\Flesh Media Server 2\logs\access.03.log' WHERE c-ip='127.0.0.1'" -i:W3C
```

Figure 18-10 shows the output that this query generates.



| LogFilename | RowNumber | x-category | x-event | date | time | x-pid | c-ip | cs-bytes | sc-bytes | x-sname | sc-stream | x-file-size | x-file-len |
|-------------|-----------|------------|------------|------------|----------|-------|-----------|----------|----------|------------|-----------|-------------|------------|
| C:\Program | 10 | session | connect | 23/11/2006 | 11:10:12 | 1760 | 127.0.0.1 | 6146 | 6146 | | | | |
| C:\Program | 11 | stream | play | 23/11/2006 | 11:10:12 | 1760 | 127.0.0.1 | 6206 | 6424 | snippet_5E | 0 | | |
| C:\Program | 12 | stream | stop | 23/11/2006 | 11:10:49 | 1760 | 127.0.0.1 | 6240 | 6802 | snippet_5E | 334 | | |
| C:\Program | 13 | session | disconnect | 23/11/2006 | 11:10:49 | 1760 | 127.0.0.1 | 6240 | 6802 | | | | |
| C:\Program | 14 | session | connect | 23/11/2006 | 11:10:50 | 1760 | 127.0.0.1 | 6146 | 6205 | | | | |
| C:\Program | 15 | stream | play | 23/11/2006 | 11:10:50 | 1760 | 127.0.0.1 | 6216 | 6424 | mystream | 0 | | |
| C:\Program | 16 | stream | stop | 23/11/2006 | 11:10:55 | 1760 | 127.0.0.1 | 6216 | 6842 | mystream | 374 | | |
| C:\Program | 17 | session | disconnect | 23/11/2006 | 11:10:55 | 1760 | 127.0.0.1 | 6216 | 6842 | | | | |
| C:\Program | 18 | session | connect | 23/11/2006 | 11:10:55 | 1760 | 127.0.0.1 | 6146 | 6205 | | | | |

Figure 18-10
The spreadsheet lists all activity generated by IP 127.0.0.1.

This type of SQL syntax and output capability makes Log Parser an immensely powerful tool. At the same time, it also poses a distinct learning curve to users new to SQL.

To find out more about this tool, note that the Log Parser installation directory contains a variety of sample files and help documents. A quick Google search also yields some interesting results.

The Power of SharedObjects

Flash Media Server does a lot more than just stream video. In this section, I will provide a brief overview of some of its capabilities and associated features so you can identify additional potential uses. Leveraging these features will help you to take your next video-on-demand project to a new level, by offering a depth and richness of features that is unparalleled by other platforms.

Author's Tip

Most of the features listed here are only available when using a dedicated Flash Media Server or shared Flash Media application hosting. They will likely be disabled when using CDN delivery for your video files. It is, however, possible to combine CDN delivery of video with dedicated Flash Media Server features inside a single application.

Use Cases

SharedObjects are, in my opinion, the coolest feature of Flash Media Server after video. SharedObjects are somewhat similar to a mini-database where your application can store information, be it related to connected users, the state of the application or pretty much anything else that you see fit.

SharedObjects are nothing new to Flash; they are available as Local SharedObjects to any Flash application and are mainly used like traditional browser cookies—for example, to store a username between visits.

When used with Flash Media Server, however, you can access Remote SharedObjects, stored on the server and accessible (if you want them to be) to all connected

clients. This may not sound like a big deal, but it offers immense possibilities. It is, for example, fairly trivial to create a text chat application using just a single SharedObject.

Whenever a user submits a message to the server, this data is stored in a Remote SharedObject and relayed back to all connected clients. This process happens virtually automatically, using a method on the SharedObject called `onSync`. This method fires automatically once the data inside the SharedObject changes, and it eliminates the need for clients to poll the server for new information. Instead, any new data is automatically sent to the connected users. My website features a few examples of this, including `coMMChat` which can be found at <http://www.flashcomguru.com/commchat.cfm>.

To take this concept a step further, you could create an application where the server queries an external data source for information—for example, stock data. As soon as the external data changes, the server can write this information to the SharedObject, which in turn propagates it to all connected clients. The application could even offer online and offline modes that would synchronize a disconnected client once they come online.

Inside the Industry



Alternatively (and where video features are less of a priority), the newly released Flex 2 Data Services, part of the Flex 2 platform, is using features derived from Flash Media Server to deliver data synchronization, data paging, publish/subscribe messaging and more to allow developers to build the next generation of rich Internet applications.

Flex 2 is a rich Internet application framework based on Flash that enables developers to create highly scalable, data-driven applications that can reach virtually anyone on any platform. For more information on Flex check out <http://www.adobe.com/products/flex/>.

I am sure you can think of many other ways to combine video and data features to enrich the user experience. A simple text chat running alongside a video can really enhance an otherwise fairly standard video player, allowing for co-browsing features and a community-like experience in which several visitors are connected on a single page.

Or why not enhance the display with a slideshow running alongside the video? It is a real bonus that Flash Media Server can deliver all this without the need for third-party technology, and that it can all be delivered within a single Flash application.

Where to Go from Here

As you probably guessed, I have barely scratched the surface of SharedObjects's capabilities. Moreover, they constitute just one of many additional Flash Media Server features and it would take another book to describe them all in detail.

To learn more about Flash Media Server and its capabilities, visit <http://www.adobe.com/products/flashmediaserver> or check out the Developer Center at <http://www.adobe.com/devnet/flashmediaserver>.

Lastly, do not hesitate to subscribe to the Flash Media List, a mailing list that is frequented by many Flash developers and video professionals and where your questions will swiftly be addressed. The address is <http://www.flashcomguru.com/flashmedialist>.

Conclusion

Flash Media Server is a complex beast, no doubt about it. But don't be intimidated by its wealth of features.

Instead, concentrate on the parts that you and your project require. If streaming Flash video delivery is your only requirement, then consider outsourcing your hosting requirements and get on with the job at hand.

Likewise, if your goal is to create more customized and integrated online experiences, then you are now armed with the basic knowledge required to get up and running with Flash Media Server and can even set up some of its more complex features.

Keep in mind, however, that this book concentrated mainly on the server's video features and that there is a multitude of other features waiting to be discovered. Do feel free to get in touch via the book's website at www.flashvideobook.com and tell us about the applications you are building.

The next chapter contains several case studies detailing fun and interesting applications of Flash video creation and distribution.