Using Flash Media Server Components

Acknowledgments

Project Management: Suzanne Smith

Writing: John Norton, Suzanne Smith

Editing: Evelyn Eldridge, Mary Ferguson, Lisa Stanziano, Anne Szabla

Production Management: Adam Barnett

Media Design and Production: Aaron Begley, Paul Benkman, John Francis, Mario Reynoso

First Edition: October 2005

Macromedia, Inc.
601 Townsend St.
San Francisco, CA 94103

Contents

Using Flash Media Server Components

This document describes the 16 media components used with Macromedia Flash Media Server 2:

Before you can be successful building your own application with components, you must be proficient in writing ActionScript for basic functionality such as connecting to the server; playing, publishing, and recording streams; and using shared objects. Components can help you develop applications quickly, but you need to understand the core concepts of Flash Media Server first. Before you use components, look through *Developing Media Applications* (included as a PDF file on the Flash Media Server CD and also available in the Flash Media Server online help).

When you start developing with components, Macromedia recommends that you read about the SimpleConnect component (see "SimpleConnect component" on page 69) before learning about the other components. You may want to work through the example in RoomList component last, because it is the most advanced and requires additional server-side scripting.

Each section discusses how to use the component and its scripting requirements (if any) and provides an example of an application you can create to test the component. Following each section on using a component is a section on the component's application programming interface (API), which describes the component's object and methods. Components are supported by Macromedia Flash Player 6 and later.

About the component framework

The components discussed in this document provide common functionality that you can use to quickly build applications with Flash Media Server. These components are built on top of the Flash Media Server component framework. You can use the framework to create your own components, or extend those included with Flash Media Server.

For more information on the component framework, see "Understanding the Component Framework" in the Macromedia Developer Center (www.macromedia.com/desdev/mx/flashcom/articles/framework.html).

Using components in a simple application

After you have set up your development environment and become familiar with Flash Media Server and basic ActionScript functions, you're ready to create an application using components. In the sample that follows, you create a simple application that uses the PeopleList component to display a list of active users.

To create the sample application:

1. Start Flash Media Server, if it is not already running.

   In Windows, select Start > Programs > Macromedia > Flash Media Server 2 > Start Flash Media Server 2.

   In Linux, open a shell window, change to the Flash Media Server installation directory, and type `./server start`. Then, to start the Admin Service, type `./adminserver start`.
2. Create a directory named com_test in the Flash Media Server applications directory.
3. In Macromedia Flash, select Window > Components.

4. From the Components panel, drag the PeopleList component onto the Stage.
5. In the Property inspector, name this instance `peopleList_mc`.
6. To connect to the server, select the first keyframe in the Timeline; in the Actions panel for the frame, provide the following code:

```
// Create a new network connection and add a stop action.
nc = new NetConnection();
stop();

// Create a status handler.
nc.onStatus = function(info) { trace(info.code); }

// optional call to clean up the list
peopleList_mc.close();

// Connect to the application and provide a user name.
nc.connect("rtmp:/com_test","Jane");
```

> **NOTE** If your Flash Media Server is not running on `localhost`, you must provide the full Uniform Resource Identifier (URI) of this test application (for example, rtmp://www.myflashcomdomain.com/com_test). For more information on the `NetConnection.connect()` method, see the Macromedia Flash Media Server documentation.

7. Use the `nc` NetConnection instance to connect the `peopleList_mc` movie clip to the server by adding the following code to the Actions panel for the first keyframe in the Timeline:

```
peopleList_mc.connect(nc);
```

8. After saving your work, you can test it by selecting Control > Test Movie.

   You might notice that your user name does not appear in the list. The PeopleList component retrieves the list of users from the server, and you have not yet added the command to retrieve server-side results. By adding a few lines of code to your server-side script file, the component can retrieve the user names from the server.
9. Create a new file in the com_test directory and name it main.asc. In this main.asc file, add the following line of code to load the components.asc file:

```
load("components.asc");
```

> **NOTE** The server must have access to the scriptlib directory to successfully load the components.asc file. For more information on loading the components.asc file, see "Creating a development environment" in *Developing Media Applications*.

10. Create an `onConnect` handler to set the user name and accept the connection from the client, as shown in the following example:

```
// Listen for the new connection to this application.
// newUserName is a parameter passed in from the client-side nc.connect
// call.
application.onConnect = function(newClient, newUserName)
{

  // Set the global user name with the user name passed into this
  // function.
  gFrameworkFC.getClientGlobals(newClient).username = newUserName;

  // Accept the connection from the user.
  application.acceptConnection(newClient);

// Note that if your application requires additional code following the
// explicit acceptConnection, you must place that code in an
// application.onConnectAccept statement (required when using
// components).
}
```

11. Save the changes to your main.asc file.

NOTE	If you run your application and then make changes to the ASC file, you need to reload your application. You can reload from the Flash Media Server management console. In Flash (making sure you are in the Flash window, not the SWF window), select Window › Other Panels › Flash Media Server Console. Log in and select com_test_definst_ from the Active Apps instances window. Click Reload App.

12. In the Flash authoring environment, select Control > Test Movie.

In the next section, you use the SimpleConnect component to further simplify the development process.

Using the SimpleConnect component

When you create applications using components, you usually add several components, and each one needs to establish a connection to the server. You can use the same network connection instance and connect all the movie clips in your application to the server. The SimpleConnect component handles all your media objects' connections. In addition, SimpleConnect provides an interface for users to log in to your application.

The following example uses the com_test application to show how SimpleConnect works. The PeopleList component does not use a NetConnection object but communicates with the server using the SimpleConnect connection. Besides making your code more efficient by managing the server connections, SimpleConnect also reduces the amount of ActionScript you need to provide. In the example, SimpleConnect handles the connection to the server and associates the PeopleList component to that connection, which eliminates the need for ActionScript. Instead of hard-coding the user name into the `nc.connect` call, you provide it at runtime.

To connect using the SimpleConnect component:

1. Make sure that Flash Media Server is running.
2. Create a directory in the Flash Media Server applications directory, and name it com_test_simcon.
3. Create a new file in com_test_simcon, and name it main.asc. Add the following line of code to load the components.asc file:

   ```
   load("components.asc");
   ```

NOTE	Alternatively, you can name your main ASC file com_test_simcon.asc.

4. In Flash, if you do not see the Components panel, select Window > Components.
5. From the Components panel, drag the PeopleList component onto the Stage.
6. In the Property inspector, name this instance of the PeopleList component `peopleList_mc`.
7. To set up a connection to the Flash Media Server, drag the SimpleConnect component onto the Stage.

8. In the Property inspector, provide the following two parameters:
   - In the Application Directory text box, type `rtmp:/com_test_simcon`. The com_test_simcon portion of the URI provides the application name and matches the directory name that you created in step 2.

     NOTE If your Flash Media Server is not running on `localhost`, you must provide the full Uniform Resource Identifier (URI) of this test application (for example, rtmp://www.myflashcomdomain.com/com_test_simcon).

   - Double-click the Components text box. In the Values dialog box that appears, click the plus sign (+), type `peopleList_mc`, and click OK.

   By providing this value, the SimpleConnect component handles the connection for the PeopleList component. When the SimpleConnect component successfully connects to the server, the PeopleList component automatically connects to the server.
9. Save and publish this test file as com_test_simcon.swf.
10. Open the SWF file, or in the Flash authoring environment, select Control > Test Movie. Log in with any user name.

    The user name you entered appears in the PeopleList component.

The SimpleConnect component provides a login interface. To increase the list of active users, open multiple instances of this application and enter a new user name for each instance.

Adding the SimpleConnect component to the first example of com_test might seem excessive because you simply dragged the PeopleList component onto the Stage, named it peopleList_mc, and added the following code:

```
nc = new NetConnection();
nc.onStatus = function(info) { trace(info.code); }
peopleList_mc.close(); // optional, cleanup call
nc.connect("rtmp:/com_test","Jane");
peopleList_mc.connect(nc);
```

The SimpleConnect component, however, is recommended and is convenient to use as your application grows. For each new component that you add, simply go to the Property inspector of the SimpleConnect component, double-click the Components text box, and add the instance name of the new component.

Without the SimpleConnect component

For most components to function properly, if you do not use the SimpleConnect component, you must add the following line of code to your server-side `application.onConnect` method to successfully register the user name with the server:

```
gFrameworkFC.getClientGlobals(newClient).username = newUserName;
```

As described in the *Server-Side ActionScript Language Reference*, whenever you create an `onConnect` method, you must explicitly accept the new client's connection.

If you don't use the SimpleConnect component, your server-side script file (main.asc or *app_name*.asc) file must include the following lines of code:

```
load("components.asc");
// Listen for the new connection to this application.
// newUserName is a parameter passed in from the client-side nc.connect
  call.
application.onConnect = function(newClient, newUserName)
{

  // Set the global user name with the user name passed into this function.
  gFrameworkFC.getClientGlobals(newClient).username = newUserName;

  // Accept the connection from the user.
  application.acceptConnection(newClient);

  // Note that if your application requires additional code following the
  // explicit acceptConnection, you must place that code in an
  // application.onConnectAccept statement (required when using components)
}
```

With the SimpleConnect component

If you use the SimpleConnect component, you need to add the following code to your main.asc file:

```
load("components.asc");
```

Using components, you have created a simple application with client-server functionality and a minimum of server-side scripting. Of course, you can use server-side scripting to enhance the functionality provided by the components and also to create your own components. Before you begin writing your own scripts, be sure to read through the conceptual information provided in Chapter 2, "Flash Media Server Architecture," in *Developing Media Applications*, which provides an important overview of the objects available to you as a Flash Media Server developer.

Advanced development with components

When you are comfortable creating media applications with components, you can take advantage of the flexibility of components to redesign UI elements, develop your own components, and use the API to enable lurker mode.

Reskinning components

When you add components to a Flash document, a Components directory is added to the Library panel. That directory contains a subdirectory named UI Components. Within that subdirectory is a skins subdirectory for each component that contains the user interface elements that you can redesign. In this document, the description of each component contains a "reskinning" section that points to the location of the component skins. You can edit the skins to change the appearance of components. You can change the skin of an instance of a component by selecting the component on the Stage, right-clicking, and selecting Edit in Place. You can also edit the component skin by double-clicking it in the directory in the Library panel.

Developing your own components

Advanced developers can create media components by creating a Flash movie clip and some server-side code. The client-side portion is implemented as a Flash movie clip and runs in a Flash client environment. The server-side portion is an ActionScript class that extends FCComponent, the component base class. For information on how to create components, see the Macromedia Developer Center (www.macromedia.com/go/fcs_components).

Understanding lurker mode

Some components provide a user mode that shields the user from other connected users. This mode is called *lurker mode*, and can be useful, for example, in scenarios that require a moderator or monitor. In lurker mode, a user is not necessarily exposed to other users and might not have access to full functionality. The Chat component, for example, supports a lurker mode that lets the lurker see other users text-chatting but prevents the lurker from sending messages. Different components provide different lurker-mode functionality. The functionality that is described in this document assumes that your application uses the SimpleConnect component, which prevents a user from logging out of an application without exiting. Each component's description tells whether lurker mode is available and describes its functionality.

AudioConference component

The AudioConference component allows you to create a *multiplexed* application; that is, it provides for the simultaneous streaming of multiple clients. This component also provides a user interface (UI) for listing users who are currently logged in to the application. The UI displays a light for each connected user; the light turns green when the user sends audio. The user clicks the Talk button to join the conversation, or selects the Auto check box, which sends the audio automatically when the user speaks. This component requires users to log in before they appear in the list and are heard by other users.

This component does not require scripting if you use it with the SimpleConnect component.

Using the AudioConference component

The following list describes several ways to use this component:

- Party line: Allow a group of users to discuss topics simultaneously.
- Virtual phone conference: Host office meetings online instead of using conference call-in numbers and speaker phones. You can use this component to easily build custom applications that use your corporate identity.
- Technical support: Along with training materials or complex applications, use this component in a problem-solving forum.

To use the component in your application:

1. Make sure that Macromedia Flash Media Server is running.
2. Create a directory in the Flash Media Server applications directory, and name it audioconf_test.

3. Create a new file in the audioconf_test directory and name it main.asc. Add the following line of code to load the components.asc file:

```
load("components.asc");
```

NOTE	You can also copy the main.asc file from another application directory to this application's directory.

4. In Flash, drag the AudioConference component onto the Stage.
5. In the Property inspector for the AudioConference component, type an instance name, such as `audioconf_mc`.
6. Drag the SimpleConnect component onto the Stage.
7. In the Property inspector for the SimpleConnect component, provide the following two parameters:
   - In the Application Directory text box, type the URI of the application that you created in step 2: `rtmp:/audioconf_test`.
   - Double-click the Components text box. In the Values dialog box that appears, click the plus sign (+) and type the instance name of the AudioConference component: `audioconf_mc`. Click OK.

   By providing this value, you are using the SimpleConnect component to handle the connection for the AudioConference component. When the SimpleConnect component successfully connects to the server, the AudioConference component automatically connects to the server.
8. Save and publish the file as myAudioApp.swf.
9. Open the SWF file. (Alternatively, select Control > Test Movie in the Flash authoring environment.) Log in and click the Auto check box.

   If your audio input device is running, the light turns green when you speak. A light appears next to each user name. If you are not speaking, your light dims.

Reskinning the AudioConference component

The assets for this component are located in the Library panel in the Core Assets - Developer Only\AudioConference Assets directory.

Related components

SimpleConnect component

FCAudioConference class

The FCAudioConference object does not require scripting if you use it with the SimpleConnect component. To appear in the list and be able to talk, a user must log in.

The FCAudioConference class provides the client-side and server-side functionality for connecting your objects to the server and cleaning up.

Method summary for the FCAudioConference class

Method	Description
`FCAudioConference.connect()`	Connects to an application on the server.
`FCAudioConference.setUsername()`	Sets the user name to display in the list of active users.
`FCAudioConference.close()`	Cleans up and removes the user from the audio conference.

FCAudioConference.connect()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

myAudioConf_mc.connect(*nc*)

Parameters

nc The active NetConnection object.

Returns

Nothing.

Description

Method; connects to an application on the server. This method also sets up all the assets needed by the component (on the client side and the server side) and calls the `setUsername()` method with the name that the server provides.

FCAudioConference.setUsername()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
myAudioConf_mc.setUsername([newName])
```

Parameters

newName A string that contains the user name to display in the list. If *newName* is not `null` or `undefined`, the user name appears in the list, and the Talk and Auto options are available. If *newName* is `null`, nothing changes, and the user is in lurker mode, which means that other users cannot detect this user.

Returns

Nothing.

Description

Method; sets the user name to display in the list of active users.

FCAudioConference.close()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
myAudioConf_mc.close()
```

Parameters

None.

Returns

Nothing.

Description

Method; cleans up after itself by freeing assets used by this component on this client and by removing the user from the audio conference.

AVPresence component

The AVPresence component lets you send and receive audio and video within your application. The component acts as a presenter seat, which any connected user can use. If a user is sending audio or video (or both) from an instance of the component, the other users see and hear that person's audio and video automatically. If an instance of the component is not being used, a user can click on it to send audio and video. The user has controls for sending and receiving as well as for disabling the audio and/or video. When a user uses the controls on an incoming stream, the effect is local; when a user uses the controls on his or her own audio and/or video instances, the user interfaces of all the other connected users are affected. In this way, each user can control whom they see and hear as well as whether they are seen and heard.

This component does not require scripting if you use it with the SimpleConnect component. You need to provide a unique instance name for each component instance that you drag onto the Stage. The AVPresence component has several component parameters that can be modified.

Using the AVPresence component

This component is a versatile media component and can be used to create a virtual presence for users. The following list describes two ways that you can use this component:

- Panel discussions: A group of people can converse before an audience and create an ongoing debate or presentation.
- Video phone: Using two of these AVPresence components in one application, you can create a video phone application.

To use the component in your application:

1. Make sure that Flash Media Server is running.
2. Create a directory in the Flash Media Server /application directory, avPresence_test.
3. Create a main.asc file in the avPresence_test directory with the following code:

```
load("components.asc");
```

> **NOTE** You can also copy the main.asc file from another application directory to this application's directory.

4. In Flash, drag the AVPresence component onto the Stage.

5. In the Property inspector, provide the following values:
   - Give the component an instance name, such as `avPresence_mc`.
   - For parameter definitions, see "AVPresence component Property inspector" on page 19.
6. Drag the SimpleConnect component onto the Stage to set up a connection to the Flash Media Server.
7. In the Property inspector for the SimpleConnect component, provide the following two parameters:
   - In the Application Directory text box, type the URI of the application that you created in step 2: `rtmp:/AVPresence_test`.
   - Double-click the Components text box. In the Values dialog box that appears, click the plus sign (+), type the instance name of the AVPresence component (for example `avPresence_mc)`, and click OK.

   By providing this value, the SimpleConnect component handles the connection for the AVPresence component. When the SimpleConnect component successfully connects to the server, the AVPresence component automatically connects to the server.
8. Save and publish the file as myAVPresence.swf.
9. Open the SWF file. (Alternatively, select Control > Test Movie in the Flash authoring environment.) Log in and click the Send Audio/Video button.

   If you have not previously set privacy settings for Flash Player, you will be asked for permission to allow camera and audio access.

   If your audio device is running, the audio level moves up when you speak. If your video device is running, you can see the output in the video display. Move your mouse pointer over the video display to see the UI for the microphone and camera as well as a button that lets you stop transmitting audio and video data.

AVPresence component Property inspector

You can set the following variables from the Property inspector for your component instance.

Name	Variable	Description
Sync Speed	`updateFps`	Number; default value: 3. Sets how many times per second this component sends messages. This value is independent of the audio and video quality, and it controls how quickly an instance responds to sending and receiving new streams as well as the update rate of the audio meter.
Video Width	`vidWidth`	Number; default value: 120. Sets the width, in pixels, to be used for the local camera setting. `vidWidth` is overwritten when FCSetBandwidth is used.
Video Height	`vidHeight`	Number; default value: 120. Sets the height, in pixels, for the local camera setting. `vidHeight` is overwritten when FCSetBandwidth is used.
Video Bandwidth	`vidBandwidth`	Number; default value: 12,000. Sets the maximum bandwidth, in bits, for the local camera setting. `vidBandwidth` is overwritten when FCSetBandwidth is used.
Video Quality	`vidQuality`	Number; default value: 0. Sets the maximum quality for the local camera setting; 0 indicates the use of the highest possible quality for the given bandwidth. `vidQuality` is overwritten when FCSetBandwidth is used.
Video FPS	`vidFps`	Number; default value: 10. Sets the frames per second for the local camera setting. `vidFps` is overwritten when FCSetBandwidth is used.

Reskinning the AVPresence component

The assets for this component are located in the Core Assets - Developer Only\AVPresence Assets directory in the Library panel.

Related components

SimpleConnect component

AVPresence class

The AVPresence class provides the client-side and server-side functionality for connecting your objects to the server and cleaning up.

Method summary for the AVPresence class

Property	Description
`FCAVPresence.connect()`	Sets up all the assets that this component needs to function (on the client side and the server side), and calls the `setUsername()` method with the name that the server provides.
`FCAVPresence.setUsername()`	Gets the user name to display in the list of active users.
`FCAVPresence.close()`	Closes the connection to the server.

FCAVPresence.connect()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

avPresence_mc`.connect(`*nc*`)`

Parameters

nc The active NetConnection object.

Returns

Nothing.

Description

Method; sets up all the assets that this component needs to function (on the client side and the server side), and calls the `setUsername()` method with the name that the server provides.

FCAVPresence.setUsername()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
avPresence_mc.setUsername(newName)
```

Parameters

newName An optional string that describes the new user name. If the *newName* parameter is not `null` or `undefined`, Flash creates a new reference for the user and sets the name to *newName*. If *newName* is `null`, nothing changes, and the user is in lurker mode.

Returns

Nothing.

Description

Method; gets the user name to display in the list of active users.

FCAVPresence.close()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
avPresence_mc.close()
```

Parameters

None.

Returns

Nothing.

Description

Method; cleans up after itself by freeing assets that this component uses on this client.

Chat component

This component shows a regular text chat window. It supports lurker mode. When you use this component with the SimpleConnect component, lurker mode prevents the display of the input text box and Send buttons if the user has never logged in. The lurker can see other people text-chatting. After the user logs in with a user name (through the SimpleConnect component or in the `nc.connect` call, as a first parameter), the user can send text. If you also use the UserColor component, each user has an individual color assigned to his or her text.

The Chat component does not require scripting if you use it with the SimpleConnect component.

You can use this component with the UserColor component. Whenever the user chooses a new color, the global color on the server side updates and the color updates for all clients.

Using the Chat component

You can use this component to create a chat room application or as a component within a larger application, as described in the following list:

- Regular chat room: Use this component with the UserColor component, the PeopleList component, and the SimpleConnect component to create a chat room.
- Useful in every application: Having a Chat component in all your applications is useful. Text communication is efficient and can provide a backup in case audio-video communication is not convenient (for example, if users' computers have slow connections).

To use the Chat component in your application:

1. Make sure that Flash Media Server is running.
2. Create an directory in the Flash Media Server applications directory, and name it chat_test.
3. Create a file named main.asc with the following code and save it in the chat_test directory:

```
load("components.asc");
```

> **NOTE** You can also copy the main.asc file from another application directory to this application's directory.

4. In Flash, drag the Chat component onto the Stage, and, in the Property inspector, give it an instance name, such as `chat_mc`.
5. Drag the UserColor component onto the Stage, and, in the Property inspector, give it an instance name, such as `color_mc`.
6. Drag the SimpleConnect component onto the Stage.

7. In the Property inspector for the SimpleConnect component, provide the following two parameters:
   - In the Application Directory text box, type the URI of the application that you created in step 2: `rtmp:/chat_test`.
   - Double-click the Components text box. In the Values dialog box that appears, click the plus sign (+), type the instance name of the Chat component (for example, `chat_mc`), click the plus sign (+) again, type the instance name of the UserColor component (for example, `color_mc`), and click OK.

   By providing these values, the SimpleConnect component handles the connection for the Chat and UserColor components. When the SimpleConnect component successfully connects to the server, the Chat and UserColor components automatically connect to the server.
8. Save and publish this file as chat_test.swf.
9. Open the SWF file or, in the Flash authoring environment, select Control > Test Movie, and log in.

   You can open multiple instances of the SWF file, log in as different users, and pick different text colors.

Reskinning the Chat component

The assets for the Chat component are located in the Core Assets - Developer Only\Chat Assets directory in the Library panel.

Related components

SimpleConnect component, UserColor component, and PeopleList component

FCChat class

The FCChat class provides the client-side functionality for connecting your objects to the server and cleaning up. You can also clear chat history from the client programmatically. For the server-side API to this component, see "FCChat server-side class" on page 26.

Method summary for the FCChat class

Method	Description
FCChat.connect()	Sets up all the assets needed by the component (on the client side and the server side) and gets the chat history.
FCChat.setUsername()	Gets the user name to display in the list of active users.
FCChat.clearHistory()	Clears the chat history and forces all the clients to update.
FCChat.close()	Cleans up after itself.

FCChat.connect()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

chat_mc.connect(*nc*)

Parameters

nc The active NetConnection object.

Returns

Nothing.

Description

Method; sets up all the assets needed by the component (on the client side and the server side) and gets the chat history.

FCChat.setUsername()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

chat_mc.setUsername(*newName*)

Parameters

newName A string.

If the *newName* parameter is not `null` or `undefined`, it shows the input text box and Send button.

If the *newName* parameter is `null`, it does nothing (lurker mode).

Returns

Nothing.

Description

Method; gets the user name to display in the list of active users.

FCChat.clearHistory()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
chat_mc.clearHistory()
```

Parameters

None.

Description

Method; clears the chat history and forces all the clients to update. This operation succeeds only if the server-side setting allows clearing by clients (this setting is enabled by default). For more information, see `FCChat.allowClear`.

FCChat.close()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
chat_mc.close()
```

Parameters

None.

Returns

Nothing.

Description

Method; cleans up after itself by freeing assets that this component uses on this client.

FCChat server-side class

The FCChat server-side class provides the server-side functionality for connecting your history properties and for enabling history clearing.

Method summary for the FCChat server-side class

Method	Description
`FCChat.clearHistory()`	Clears the history.

Property summary for the FCChat server-side class

The following server-side properties can be set to modify the behavior of the component.

Property	Description
`FCChat.histlen`	Defines how large a chat history can get before it truncates.
`FCChat.persist`	Defines whether a chat history is persistent across application lifetime.
`FCChat.allowClear`	Defines whether a chat history can be cleared from the client.

FCChat.histlen

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
FCChat.prototype.histlen=maxHistory;
```

Description

Property; defines how large a chat history can get before it truncates. This is a read/write property. The value *maxHistory* is a number that indicates the maximum length of text messages that the server retains. The default is 250.

FCChat.persist

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

FCChat.prototype.persist=*true*

Description

Property; defines whether a chat history is persistent across application lifetime. Persistent histories endure application restarts. This is a read/write property. The values `true` and `false` are Boolean values that determine whether the server saves the history. The default value is `true`.

FCChat.allowClear

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

FCChat.prototype.allowClear=*true*

Description

Property; defines whether a chat history can be cleared from the client. This is a read/write property. The values `true` and `false` are Boolean values that determine whether the user can clear the history. The default value is `true`.

FCChat.clearHistory()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
FCChat.prototype.clearHistory()
```

Parameters

None.

Description

Method; clears the history. Unlike the client side, calling this method on the server side always succeeds.

ConnectionLight component

This component provides visual feedback on the state of the client connection. It is green when connected, red when disconnected, and yellow if the *latency* (the time it takes to send data across the network connection to and from the server) of the connection is too high. The light doubles as a button that toggles a display box that provides detailed information about the connection (the data latency rate and the instantaneous upload and download rates).

The ConnectionLight component does not require scripting if you use it with the SimpleConnect component.

Using the ConnectionLight component

This simple, useful component can be used in any application. It is a standard component for all your media applications.

The ConnectionLight component is simple to use. Drag it onto the Stage and make sure components.asc is loaded in the server script. If you use this component with the SimpleConnect component, no other steps are necessary. Without the SimpleConnect component, you need to add the following line of client-side ActionScript:

```
light_mc.connect(nc);
```

In this ActionScript, `light_mc` is the instance name of the light that is being dragged into the application, and `nc` is a NetConnection object.

To use the component in your application:

1. Make sure Flash Media Server is running.
2. Create a directory in the Flash Media Server applications directory, and name it connect_test.

3. Create a file named main.asc in the connect_test directory with the following code:

```
load("components.asc");
```

> **NOTE** You can also copy the main.asc file from another application directory to this application's directory.

4. In Flash, if you do not see the Components panel, select Window > Components.
5. From the Components panel, drag the ConnectionLight component onto the Stage. In the Property inspector, name this instance `light_mc`. For additional parameters, see "ConnectionLight component Property inspector" on page 30.
6. To connect to the server, select the first keyframe in the Timeline, and in the Actions panel for the frame, provide the following code:

```
nc=new NetConnection();
nc.connect("rtmp:/connect_test");
```

7. Use the `nc` NetConnection instance to connect the `light_mc` movie clip to the server, as shown in the following example:

```
light_mc.connect(nc);
```

8. Save and publish this test file as connect_test.swf.
9. Open the SWF file in the connect_test directory or, in the Flash authoring environment, select Control > Test Movie. Click the green button.

   You should see your connection values.

Reskinning the ConnectionLight component

The assets for this component are located in the Core Assets - Developer Only\ConnectionLight Assets and ConnectionLight Assets copy directories in the Library panel.

ConnectionLight component Property inspector

You can set the following variables from the Property inspector for your component instance.

Property	Description
`measurementInterval`	This property controls how often bandwidth and latency measurements are performed. The default is 2 seconds.
`latencyThreshold`	This property determines the latency beyond which the light will turn yellow. The default is 0.1 seconds; therefore, whenever the connection latency becomes greater than 100 milliseconds, the light turns yellow and stays that way until the latency falls below the threshold.

Related components

SimpleConnect component

FCConnectionLight class

The FCConnectionLight class provides the client-side and server-side functionality for connecting your objects to the server and cleaning up. It also enables you to set the length of time it takes to send data to and from the server.

Property summary for the FCConnectionLight class

Property	Description
`FCConnectionLight.measurementInterval`	Controls how often bandwidth and latency are measured.
`FCConnectionLight.latencyThreshold`	Determines the latency beyond which the light turns yellow.

Method summary for the FCConnectionLight object

Method	Description
`FCConnectionLight.connect()`	Begins monitoring the state of the connection.
`FCConnectionLight.close()`	Stops monitoring the connection and cleans up after itself.

FCConnectionLight.measurementInterval

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
connectLight_mc.measurementInterval = maxInterval
```

Description

Property; controls how often bandwidth and latency are measured. This is a read/write property. The value *maxInterval* is a number that determines how often bandwidth and latency are measured. The default is 2 seconds.

FCConnectionLight.latencyThreshold

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
connectLight_mc.latencyThreshold = maxLatency
```

Description

Property; determines the latency beyond which the light turns yellow. This is a read/write property. The value *maxLatency* determines the length of time it takes for the light to turn yellow after the latency becomes too high. The default is 0.1 seconds; therefore, whenever the connection latency becomes greater than 100 milliseconds, the light turns yellow and stays that way until the latency falls below the threshold.

FCConnectionLight.connect()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
connectLight_mc.connect(nc)
```

Parameters

nc The active NetConnection object.

Returns

Nothing.

Description

Method; begins monitoring the state of the connection.

FCConnectionLight.close()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
connectLight_mc.close()
```

Parameters

None.

Returns

Nothing.

Description

Method; cleans up after itself by freeing assets that this component uses on this client and by no longer monitoring the connection.

Cursor component

This component shows a mouse pointer (cursor) for each user who is connected to an application. As each user moves the mouse within the application, the movement is reflected on all other users' screens. Users' names appear next to their own pointer, and users can specify a color for their pointer.

The Cursor component does not require scripting if you use it with the SimpleConnect component.

You can use this component with the UserColor component.

NOTE

When this component is part of a media application and the user first moves the pointer over the Stage, no pointer is visible. After the user logs in, the user's pointer appears, as well as the pointers of any other logged-in users.

Using the Cursor component

This component is a simple implementation of an *avatar*—an image that represents a user on a virtual, multiuser Stage. When you implement this component, each pointer on the Stage represents one user's position in the media application.

The following list describes several ways to use this component:

- Educational applications: Drop this component into an e-learning application to enable all your users to point out parts of graphs, maps, or formulas during presentations, discussions, or questions.
- Usability studies: Add this component to your application (for example, your website) to see how people use it.
- Visual help for new interaction models: In multiuser applications, seeing the locations of all mouse pointers on the Stage gives each user a better sense of what other users are doing.

To use the component in your application:

1. Make sure Flash Media Server is running.
2. Create a directory in the Flash Media Server applications directory and name it cursor_test.
3. In the cursor_test directory, create a file named main.asc with the following code:

```
load("components.asc");
```

NOTE

You can also copy the main.asc file from another application directory to this application's directory.

4. Drag the Cursor component onto the Stage.
5. In the Property inspector, give the component an instance name, such as `cursor_mc`.
6. Drag the SimpleConnect component onto the Stage.

7. In the Property inspector for the SimpleConnect component, provide the following two parameters:
   - In the Application Directory text box, type the URI of the application that you created in step 2: `rtmp:/cursor_test`.
   - Double-click the Components text box. In the Values dialog box that appears, click the plus sign (+) and type the instance name of the Cursor component: `cursor_mc`. Click OK.

   By providing this value, you are using the SimpleConnect component to handle the connection for the Cursor component. When the SimpleConnect component successfully connects to the server, the Cursor component automatically connects to the server.
8. Save and publish the file as cursor_test.swf.
9. Open the SWF file. (Alternatively, select Control > Test Movie in the Flash authoring environment.) Log in and click the Auto check box.

   You see your pointer with your login name. If you open more instances of the application, you will see additional pointers.

Reskinning the Cursor component

The assets for this component are located in the Core Assets - Developer Only\Cursor Assets directory in the Library panel. You can edit the pointer arrow and label text to customize your application.

Related components

SimpleConnect component and UserColor component

FCCursor class

The FCCursor class provides the client-side and server-side functionality for connecting your objects to the server and cleaning up. It also allows you to set the user's pointer color programmatically.

Method summary for the FCCursor class

Method	Description
FCCursor.connect()	Sets up all the assets needed by the component (on the client side and the server side) and calls setUsername with the name that the server provides.
FCCursor.setUsername()	Sets the user name to display and pass to the server.
FCCursor.close()	Cleans up and shows the normal mouse pointer by calling the Mouse.show() method.
FCCursor.setColor()	Changes the color of the user's pointer.

FCCursor.connect()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

cursor_mc.connect(*nc*)

Parameters

nc The active NetConnection object.

Returns

Nothing.

Description

Method; sets up all the assets needed by the component (on the client side and the server side) and calls setUsername() with the name that the server provides.

FCCursor.setUsername()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

cursor_mc.setUsername(*newName*)

Parameters

newName An optional string that specifies the new user name. If the *newName* parameter is not `null` or `undefined`, the method creates a new reference for the user, sets the name of the reference to *newName*, and hides the normal mouse pointer by calling the `Mouse.hide()` method. If *newName* is `null`, nothing changes and the user is in lurker mode.

Returns

Nothing.

Description

Method; sets the user name to display and pass to the server.

FCCursor.close()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
cursor_mc.close()
```

Parameters

None.

Returns

Nothing.

Description

Method; cleans up after itself by freeing assets used by this component on this client, and returns the pointer to its previous state by calling the `Mouse.show()` method.

FCCursor.setColor()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
cursor_mc.setColor(newColor)
```

Parameters

newColor A string that indicates a hexadecimal color (such as 0xFC00F3).

Returns

Nothing.

Description

Method; changes the color of the user's pointer to the specified color. The Cursor component, which can be used with the UserColor component, is designed to monitor changes to the value gFlashCom.usercolor. A change to this value causes setColor() to be called, and updates the colors for all clients.

PeopleList component

This component displays a list of users who are connected to a media application. Only users who have supplied a user name appear in the list.

This component does not require any scripting when you use it with the SimpleConnect component.

Using the PeopleList component

This component is standard in most media applications. It provides a list of currently logged-in users.

The following list describes two ways that you can use this component:

- Chat: Use the PeopleList component with the Chat and SimpleConnect components to create the basis for a chat room application.
- Virtual meetings: Use this component when creating a virtual conference room to display all the users who are currently participating in the meeting.

Reskinning the PeopleList component

The assets for this component are located in the Core Assets - Developer Only\PeopleList Assets directory in the Library panel.

Related components

SimpleConnect component and PeopleList component

FCPeopleList class

The FCPeopleList class provides the client-side and server-side functionality for connecting your objects to the server and cleaning up. It also allows you to set the lurker mode programmatically.

Variables

Name	Variable	Description
Lurkers	`lurkers`	Integer; this variable updates whenever a new user connects or disconnects. It has a value that is equal to the number of users who are connected to the application but have not supplied a user name. This variable should only be read and not set.
Users	`users`	Integer; this variable is similar to *`lurkers`*. It has a value that is equal to the number of users who are connected to the application and have supplied a user name. This variable should only be read and not set.

Method summary for the FCPeopleList class

Method	Description
`FCPeopleList.connect()`	Sets up all the assets needed by the component (on the client side and the server side), and calls the `setUsername()` method with the name that the server provides.
`FCPeopleList.setUsername()`	Sets the user name that appears in the list of active users.
`FCPeopleList.close()`	Cleans up after itself.

FCPeopleList.connect()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

`peopleList_mc``.connect(`*`nc`*`)`

Parameters

nc The active NetConnection object.

Returns

Nothing.

Description

Method; sets up all the assets needed by the component (on the client side and the server side), and calls the `setUsername()` method with the name that the server provides.

FCPeopleList.setUsername()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

peopleList_mc`.setUsername(`*newName*`)`

Parameters

newName An optional string that describes the new user name. If *newName* is not `null` or `undefined`, *newName* appears in the list of users. If *newName* is `null`, nothing changes and the user is in lurker mode.

Returns

Nothing.

Description

Method; sets the user name that appears in the list of active users.

FCPeopleList.close()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

peopleList_mc`.close()`

Parameters

None.

Returns

Nothing.

Description

Method; cleans up after itself by freeing assets that this component uses on this client and by removing the user from the list of users in the client.

PresentationSWF component

The PresentationSWF component allows you to create a presentation by showing a shared version of another SWF file. The component can act in two modes: in *speaker mode* the user is the presenter and controls the SWF file so all users see the same frame simultaneously; in *default mode* users are viewers and can navigate the SWF file by using Next and Back buttons to view the file asynchronously, but cannot progress past the point in the presentation that the speaker has reached.

The PresentationSWF component does not require scripting if you use it with the SimpleConnect component.

Setting up the presentation SWF file

This component requires another SWF file to load as the presentation SWF. In the following example, you use a copy of the simple_preso.swf file, in which you can find the samples/sample_broadcast directory where you installed Flash Media Server.

When you create a presentation to display in this component, you must set up the SWF file properly:

- The file must be structured so that each frame on the main Timeline represents one presentation slide.
- Make sure there is a `stop()` statement on the first frame.
- Do not include any ActionScript in the presentation SWF file that could affect the playback on the main Timeline, which contains the component.

Using the PresentationSWF component

The following list describes two ways to use this component:

- Sales presentation: Display charts and other information while you deliver an online presentation.
- Meeting agenda: Direct users' attention to different agenda items during a meeting.

In the following example, you create two client files: presentSWF_test_speaker is for the presenter who has control of the slides, and presentSWF_test is for a viewing participant who does not have speaker capabilities. Although both of these clients point to the same application, presentSWF_test, only clients that are in speaker mode have the added speaker capabilities.

To use the component in your speaker application:

1. Make sure Flash Media Server is running.
2. Create a directory in the Flash Media Server applications directory, and name it presentSWF_test.
3. In the presentSWF_test directory, create a file named main.asc with the following code:

   ```
   load("components.asc");
   ```

   NOTE: You can also copy the main.asc file from another application directory to this application's directory.

4. In Flash, drag the PresentationSWF component onto the Stage and, in the Property inspector, give it an instance name, such as `presentSWF_mc`.
5. In the Property inspector, don't change the name of the default presentation SWF file, simple_preso.swf. To set additional parameters, see "PresentationSWF component Property inspector" on page 42.
6. Drag the SimpleConnect component onto the Stage.
7. In the Property inspector for the SimpleConnect component, provide the following two parameters:
   - In the Application Directory text box, type the URI of the application that you created in step 2: `rtmp:/presentSWF_test`.
   - Double-click the Components text box. In the Values dialog box that appears, click the plus sign (+) and type the instance name of the PresentationSWF component: `presentSWF_mc`. Click OK.

By providing this value, you are using the SimpleConnect component to handle the connection for the PresentationSWF component. When the SimpleConnect component successfully connects to the server, the PresentationSWF component automatically connects to the server.

8. In the first keyframe's Actions panel, add the following code to stop the progression of the application and to set the mode to speaker mode:

```
stop();
_global.speakerMode=true;
```

9. Save and publish this file as presentSWF_test_speaker.swf.

To use the PresentationSWF component in your viewer application:

1. Make a copy of the presentSWF_test_speaker.fla file and name it presentSWF_test.fla.
2. In presentSWF_test.fla, change the following code after the `stop` call in the first keyframe's Actions panel so that the user is not the speaker:

```
_global.speakerMode=false;
```

3. Save the presentSWF_test.fla file.

To test the PresentationSWF component in both modes:

1. Open the presentSWF_test_speaker.swf file or, in the Flash authoring environment, select Control > Test Movie, and log in.
2. Open the presentSWF_test.swf file or, in the Flash authoring environment, select Control > Test Movie, and log in as a different user.

   In presentSWF_test_speaker.swf, you are a presenter: you can create new slides and click back and forth through the presentation. You also have an outline of the presentation.

   In presentSWF_test.swf, you are a viewer: you can view the slides, but only as far as the presenter has progressed in the presentation.

PresentationSWF component Property inspector

You can set the following variables from the Property inspector for your component instance.

Name	Variable	Description
Presentation SWF	`swfFile`	String; the default value is `simple_preso.swf`. Specifies the file to load for the presentation. You can change this file at runtime by calling the `loadSWF()` method.
Viewer Buttons Enabled	`viewerButtons`	A Boolean value; the default value is `true`. If `true`, users who are not in speaker mode can navigate the presentation file independently.

Reskinning the PresentationSWF component

The assets for this component are located in the Core Assets - Developer Only\PresentationSWF Assets directory in the Library panel.

Related components

SimpleConnect component

FCPresentationSWF class

The FCPresentationSWF class provides the client-side and server-side functionality for connecting your objects to the server and cleaning up. It also allows you to manipulate the frames programmatically.

Method summary for the FCPresentationSWF class

Method	Description
`FCPresentationSWF.connect()`	Sets up all the assets needed by the component (on the client side and the server side).
`FCPresentationSWF.close()`	Cleans up and disconnects the component.
`FCPresentationSWF.loadSWF()`	Optionally used by the speaker to set a new SWF file to be used for the presentation.
`FCPresentationSWF.next()`	Advances the presentation SWF by one frame.
`FCPresentationSWF.back()`	Moves back one frame in the presentation SWF.
`FCPresentationSWF.sync()`	Resynchronizes the viewer's frame with the speaker's frame.

Variable

Global variable	Description
`_global.speakerMode`	A Boolean value; the default value is `true`. If `true`, the user can change the current slide that other users view. If `false`, users can view only the slide that the speaker chooses to display. To set this variable programmatically, set `_global.speakerMode=[true or false]`.

_global.speakerMode

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
_global.speakerMode=true
```

Description

Global variable; a Boolean value that controls speaker mode. If `true` (the default), the user can change the current slide that other users view. If `false`, users can view only the slide that the speaker chooses to display. To set this variable, set `_global.speakerMode` to `true` or `false`.

FCPresentationSWF.connect()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

presentSWF_mc.connect(*nc*)

Parameters

nc The active NetConnection object.

Returns

Nothing.

Description

Method; sets up all the assets needed by the component (on the client side and the server side).

FCPresentationSWF.close()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
presentSWF_mc.close()
```

Parameters

None.

Returns

Nothing.

Description

Method; cleans up after itself by freeing assets used by this component on this client.

FCPresentationSWF.loadSWF()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
presentSWF_mc.loadSWF(swfFile)
```

Parameters

swfFile A string that specifies a relative or absolute path to a SWF file.

Returns

Nothing.

Description

Method; optionally used by the speaker to set a new SWF file to be used for the presentation. This method is not used by viewers to load a file; the SWF loads automatically when a speaker loads a new file.

FCPresentationSWF.next()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
presentSWF_mc.next()
```

Parameters

None.

Returns

Nothing.

Description

Method; advances the presentation SWF by one frame. In speaker mode, the frame advances for all synchronized viewers. If the user is not in speaker mode and the viewer buttons are enabled, a viewer can advance the frame locally (which does not affect other viewers) but cannot go past the speaker's current position.

FCPresentationSWF.back()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
presentSWF_mc.back()
```

Parameters

None.

Returns

Nothing.

Description

Method; moves back one frame in the presentation SWF. If the user is in speaker mode, the frame goes back one position for all synchronized viewers. If the user is not in speaker mode and the viewer buttons are enabled, the frame can be changed only locally, which does not affect other viewers.

FCPresentationSWF.sync()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
presentSWF_mc.sync()
```

Parameters

None.

Returns

Nothing.

Description

Method; resynchronizes the viewer's frame with the speaker's frame.

PresentationText component

This component allows you to create a shared text presentation. Like the PresentationSWF component, the PresentationText component can act in speaker mode or default mode. Speaker mode allows users to edit slides in real time, change the current slide, and add or remove slides. Default mode allows users to view the presentation but not progress past the point the speaker has reached.

The PresentationText component does not require scripting if you use it with the SimpleConnect component.

Using the PresentationText component

This component can add supplemental material to any presentation application.

The following list describes two ways to use this component:

- Training seminars: Make notes and outline examples in a training application.
- Meeting notes: Keep all users current with issues and topics in a meeting application.

In the following example, you create two client files: presentText_test_speaker is for the speaker who has control of the slides and an outline for the presentation, and presentText_test is for a viewing participant who does not have speaker capabilities. Whereas both of these clients point to the same application, presentText_test, only clients that are in speaker mode have the added speaker capabilities.

To use the PresentationText component in your speaker application:

1. Make sure Flash Media Server is running.
2. Create a directory in the Flash Media Server /application directory, and name it presentText_test.

3. In the presentText_test directory, create a file named main.asc with the following code:

```
load("components.asc");
```

NOTE	You can also copy the main.asc file from another application directory to this application's directory.

4. In Flash, drag the PresentationText component onto the Stage and, in the Property inspector, give it an instance name, such as `presentText_mc`.

NOTE	The only property in the Property inspector is Speaker Mode, and the default setting is `true`. This enables speaker capabilities for this client.

5. Drag the SimpleConnect component onto the Stage.
6. In the Property inspector for the SimpleConnect component, provide the following two parameters:
   - In the Application Directory text box, type the URI of the application that you created in step 2: `rtmp:/presentText_test`.
   - Double-click the Components text box. In the Values dialog box that appears, click the plus sign (+) and type the instance name of the PresentationText component: `presentText_mc`. Click OK.

   By providing this value, you are using the SimpleConnect component to handle the connection for the PresentationText component. When the SimpleConnect component successfully connects to the server, the PresentationText component automatically connects to the server.
7. Save and publish the speaker file as presentText_test_speaker.fla.

To use the PresentationText component in your viewer application:

1. Make a copy of the presentText_test_speaker.fla file and name it presentText_test.fla.
2. In presentText_test.fla, select the PresentationText component, and in the Property inspector, change the Speaker Mode parameter to `false`.
3. Save the presentText_test.fla file.

To test the PresentationText component in both modes:

1. Open the presentText_test_speaker.swf file or, in the Flash authoring environment, select Control > Test Movie, and log in.
2. Open the presentText_test.swf file or, in the Flash authoring environment, select Control > Test Movie, and log in as a different user.

   In presentText_test_speaker.swf, you are a speaker: you can create new slides and click back and forth through the presentation. You also have an outline of the presentation.

In presentText_test.swf, you are a viewer: you can view the slides, but only as directed by the speaker.

PresentationText component Property inspector

You can set the following variables from the Property inspector for your component instance.

Name	Variable	Description
Speaker Mode	`speakerMode`	A Boolean value; the default value is `true`. If `true`, the user can add, edit, or remove slides. The user can also change the current slide that other users view. If `false`, users can view only the slide that the speaker chooses to display. To set this variable programmatically, call `_global.speakerMode=[true or false]`.

Reskinning the PresentationText component

The assets for this component are located in the Core Assets - Developer Only\PresentationText Assets directory in the Library panel.

Related components

SimpleConnect component

FCPresentationText class

The FCPresentationText class provides the client-side and server-side functionality for connecting your objects to the server and cleaning up. It also allows you to manipulate the slides programmatically.

Method summary for the FCPresentationText class

Method	Description
`FCPresentationText.connect()`	Sets up all the assets needed by the component (on the client side and the server side).
`FCPresentationText.setUsername()`	Sets the user name to display and pass to the server.
`FCPresentationText.close()`	Cleans up and disconnects the component.
`FCPresentationText.nextSlide()`	Goes to the next slide in the list.

Method	Description
FCPresentationText.backSlide()	Goes to the previous slide in the list.
FCPresentationText.newSlide()	Creates a new slide and adds it to the list of slides.
FCPresentationText.deleteSlide()	Removes the current slide.

FCPresentationText.connect()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

presentText_mc.connect(*nc*)

Parameters

nc The active NetConnection object.

Returns

Nothing.

Description

Method; sets up all the assets needed by the component (on the client side and the server side).

FCPresentationText.setUsername()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

presentText_mc.setUsername(*newName*)

Parameters

newName An optional string that specifies the new user name. If *newName* is not `null` or `undefined`, all features of the component are made available to the user. If *newName* is `null`, nothing changes and the user is in lurker mode.

Returns

Nothing.

Description

Method; sets the user name to display and pass to the server.

FCPresentationText.close()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
presentText_mc.close()
```

Parameters

None.

Returns

Nothing.

Description

Method; cleans up after itself by freeing assets used by this component on this client and by disconnecting this component.

FCPresentationText.nextSlide()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
presentText_mc.nextSlide()
```

Parameters

None.

Returns

Nothing.

Description

Method; goes to the next slide in the list.

FCPresentationText.backSlide()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
presentText_mc.backSlide()
```

Parameters

None.

Returns

Nothing.

Description

Method; goes to the previous slide in the list.

FCPresentationText.newSlide()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
presentText_mc.newSlide()
```

Parameters

None.

Returns

Nothing.

Description

Method; creates a new slide and adds it to the list of slides.

FCPresentationText.deleteSlide()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
presentText_mc.deleteSlide()
```

Parameters

None.

Returns

Nothing.

Description

Method; removes the current slide.

RoomList component

The RoomList component is an advanced component that lets the user create, join, and delete rooms, such as chat applications. This component is integral to lobby applications, which are useful for managing user access to other applications.

The user can click the Create Room option to display a pop-up menu containing the room name and description fields. Selecting a room and clicking Join opens a new browser window and the selected room—that is, an instance of another application. The user can delete rooms from the list if no one is using them. To be able to use all the features of the component, the user must be connected to the server with a user name.

Using the RoomList component

For this component, you'll create two applications: a lobby and a room. The lobby application acts as a starting point for the user; the room acts as a destination for the user.

NOTE	You must host and access this application using a web server, either locally or remotely. For example, a typical development environment hosts test applications locally on http://localhost.

The RoomList component can launch any media application as a room. You can have several instances of the same application open, as long as each instance has a unique session.

The RoomList component does not require client-side scripting if you use it with the SimpleConnect component. The lobby application you'll create requires a main.asc file that loads the component.asc file.

The room application requires a server-side script in the main.asc file that loads the component.asc file, gets the user and room name, and manages the room and user counters. This main.asc file contains server-to-server calls to pass parameters back and forth, but none of these calls controls whether the user is actually connected to the room application. The server-side functionality simply updates the number of people displayed in each room for the lobby application and passes the name of the application to the application. This allows the room name to be displayed dynamically in each instance.

The room application that you open from the RoomList component must use the `username` and `appInstance` parameters being sent to it. If the application uses the SimpleConnect component, this process is handled automatically. For an example, see the sample_room application, which is installed with Flash Media Server.

In the lobby application you'll take the following steps:

1. Add a SimpleConnect component.
2. Add a RoomList component.
3. Load the components.asc file to enhance the lobby application.

Then, in the room application you'll take these additional steps:

1. Alter the preexisting chat room application to make it a destination point.
2. Alter the HTML wrapper to pass values from the lobby application to the chat room.

The following list describes two ways to use this component:

- Chat lobby: Create a central lobby or meeting place for users to congregate. Later, they can select a room to join.
- Application instance launcher: Create a virtual center from which users can launch multiple instances of your application.

NOTE

The following example assumes you have completed the example in "Using the Chat component" on page 22.

To use the component in a lobby application:

1. Make sure Flash Media Server is running.
2. Create a directory in the Flash Media Server applications directory, and name it lobby_com_test.
3. In Flash, drag the SimpleConnect component onto the Stage.

4. In the Property inspector for the SimpleConnect component, provide the following parameters:
   - In the Application Directory text box, type the URI of the application that you created in step 2: `rtmp:/lobby_com_test`.
   - Double-click the Components text box. In the Values dialog box that appears, click the plus sign (+) and type the instance name of the RoomList component: `roomList_mc`. Click OK.
5. Drag the RoomList component onto the Stage.
6. In the Property inspector for the RoomList component, provide the following values:
   - Give the component the instance name `roomList_mc`. This value will be used in the room application's main.asc file.
   - In the Room Application Path text box, type the relative location of the room application, for example, `chat_room_test.html`. This provides the room application HTML filename, which you will generate when you publish the chat_room_test.fla file.
7. Save and publish the file as lobby_com_test.swf.
8. In the lobby_com_test directory, create a file named main.asc with the following code:

```
load("components.asc");
```

To create a room application that works as a destination of the lobby application:

1. Make a copy of the chat_test application you created earlier in "Using the Chat component" on page 22. Rename the registered application's directory on your web server chat_room_test, and rename the FLA file copy chat_room_test.fla.
2. In the chat_room_test.fla file, select the SimpleConnect component on the Stage and change the Application Directory setting to rtmp:/chat_room_test.
3. In the Property inspector for the SimpleConnect component, give the component the instance name `connector_mc`. This value will be called from the destination room's main.asc file.
4. Save and publish the file as chat_room_test.swf.

   When you publish your application, Flash generates a chat_room_test.html file.
5. Open the chat_room_test.html file in an HTML editor and replace the `<BODY>...</BODY>` tag, and everything within it, with the following code:

```
<BODY bgcolor="#FFFFFF">
<SCRIPT LANGUAGE=JavaScript1.1>
<!--
  // This extracts any parameters passed to the HTML document
```

```
    // and passes them directly to the Flash movie.
    var appURL = String(document.location);
    if (appURL.indexOf("?") != -1) {
      var appParams = appURL.substr(appURL.indexOf("?"));
    } else {
      var appParams = "";
    }
    document.write('<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-
    444553540000"');
    document.write('  codebase="http://download.macromedia.com/pub/
    shockwave/cabs/flash/swflash.cab#version=6,0,0,0" ');
    document.write(' ID="chat_room_test" WIDTH="550" HEIGHT="300"
    ALIGN="">');
    document.write(' <PARAM NAME=movie VALUE="chat_room_test.swf' +
    appParams + '"> ');
    document.write(' <PARAM NAME=quality VALUE=high> <PARAM NAME=bgcolor
    VALUE=#FFFFFF> ');
    document.write(' <EMBED src="chat_room_test.swf' + appParams + '"
    quality=high bgcolor=#FFFFFF  ');
    document.write(' swLiveConnect=FALSE WIDTH="550" HEIGHT="300"
    NAME="chat_room_test" ALIGN=""');
    document.write(' TYPE="application/x-shockwave-flash"
    PLUGINSPAGE="http://www.macromedia.com/go/getflashplayer">');
    document.write(' </EMBED>');
    document.write(' </OBJECT>');
  //-->
  </SCRIPT><NOSCRIPT>This application requires JavaScript support</
    NOSCRIPT>
  </BODY>
```

NOTE The HTML extracts the parameters from the URL and passes them to the room application. This works properly only when you view the pages from a web server, such as http://localhost. Also, the element values must match the destination room's application and SWF filename. In the HTML above, for example, you use `chat_room_test` and `chat_room_test.swf`. If you publish your SWF file again from Flash, you will overwrite this HTML file.

6. Next, you'll add the code to the chat_room_test application's main.asc file to make it a destination for the lobby application.

NOTE The main.asc file in the chat_room_test directory should already contain the following server-side ActionScript:

```
// Load the utility file.
load("components.asc");
```

In the main.asc file of the chat_room_test application, write an `onConnect` handler that adds the user to the global list of clients, accepts the user connection, and makes the call to get the room instance name and update the room count.

```
application.onConnect = function(newClient, username, password) {

  // Save the user name.
  gFrameworkFC.getClientGlobals(newClient).username = username;

  // Accept the new user's connection.
  application.acceptConnection(newClient);

  // Get the instance name and update the room count.
  if (this.name.indexOf("/") != -1) {
    newClient.room = this.name.substr(this.name.lastIndexOf("/")+1);
    roomConnect(newClient);
  }
}
```

7. Write a `roomResult` handler to receive the result of the `roomConnect` call, and pass the result to the SimpleConnect component.

```
function roomResult(newClient) {
  this.onResult = function(roomName) {
    // "connector_mc" must match the instance name
    // of the SimpleConnect component.
    newClient.call("FCSimpleConnect/connector_mc/roomName",
                   null,
                   roomName);
  }
}
```

8. Connect to the lobby to retrieve the room name the user selected, and call `roomResult` with the return value.

```
function roomConnect (newClient, room) {

  // Create a new NetConnection.
  lobby_nc = new NetConnection();

  // If the connection to the lobby is successful,
  // make a server-to-server call, passing the room
  // that the user just joined as a parameter. This call
  // updates the number of users displayed in the lobby
  // and returns the name of the room back to the result object.
  lobby_nc.onStatus = function (infoStatus) {
    if (infoStatus.code == "NetConnection.Connect.Success") {
      // "roomlist_mc" must match the instance name of
      // the RoomList component.
      lobby_nc.call("FCRoomList/roomlist_mc/roomConnect",
                    new roomResult(newClient),
```

```
                        newClient.room);
      }
    };
    // Connect to the lobby application.
    lobby_nc.connect("rtmp://localhost/lobby_com_test");

}
```

9. Write an `onDisconnect` handler that notifies the lobby application.

```
// When the user disconnects, tell the lobby application.
application.onDisconnect = function(client) {
  if (client.room != null) {
    roomDisconnect(client.room);
  }
}
```

10. Adjust the room count and remove the user from the display.

```
function roomDisconnect (room) {

  // Create a new NetConnection.
  lobby_nc = new NetConnection();

  // If the connection is successful, make a server-to-server call
  // that will decrement the number of users displayed in the specified
  // room.
  lobby_nc.onStatus = function (infoStatus) {
    if (infoStatus.code == "NetConnection.Connect.Success") {
      // "roomlist_mc" must match the instance name of
      // the RoomList component.
      lobby_nc.call("FCRoomList/roomlist_mc/roomDisconnect", null,
room);
    }
  }

  // Connect to the lobby application.
  lobby_nc.connect("rtmp://localhost/lobby_com_test");
}
```

11. Save main.asc and reload your application.

To test the RoomList component:

1. Open the lobby_com_test.html file from a web browser and log in.
2. Select Create Room to add a room to the room list.
3. Click the room you created and click Join Room.

   An instance of the chat_room_test application opens in the browser.

4. You can chat and change your text color.
5. Open another instance of the lobby_com_test application and log in as a different user to simulate a multiuser chat session.

RoomList component Property inspector

You can set the following variables from the Property inspector for your component instance.

Name	Variable	Description
Room Application Path	`roomPath`	String; provides the path to the application that launches when you join a room. The path can be absolute or relative.

Reskinning the RoomList component

The assets for this component are in the Core Assets - Developer Only\RoomList Assets directory in the Library panel.

Related components

SimpleConnect component

FCRoomList class

The FCRoomList class provides the client-side and server-side functionality for connecting your objects to the server and cleaning up. It also allows you to manipulate the rooms programmatically.

Method summary for the FCRoomList class

Method	Description
`FCRoomList.connect()`	Sets up all the assets needed by the component (on the client side and the server side) and calls the `setUsername()` method with the name that the server provides.
`FCRoomList.setUsername()`	Sets the user name to display and pass to the server.
`FCRoomList.close()`	Cleans up and disconnects the component.
`FCRoomList.createRoom()`	Opens a Create Room dialog box and temporarily disables the Join, Create, and Delete buttons.

Method	Description
FCRoomList.deleteRoom()	Calls the server that is requesting to delete the currently selected room.
FCRoomList.joinRoom()	Launches a new browser window to open an instance of the selected room.

FCRoomList.connect()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
roomList_mc.connect(nc)
```

Parameters

nc The active NetConnection object.

Returns

Nothing.

Description

Method; sets up all the assets needed by the component (on the client side and the server side) and calls the `setUsername()` method with the name that the server provides.

FCRoomList.setUsername()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
roomList_mc.setUsername(newName)
```

Parameters

newName An optional string that specifies the new user name. If *newName* is not `null` or `undefined`, the user can see the list of rooms but cannot join, create, or delete a room. If *newName* is `null`, nothing changes and the user is in lurker mode.

Returns

Nothing.

Description

Method; sets the user name to display and pass to the server.

FCRoomList.close()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
roomList_mc.close()
```

Parameters

None.

Returns

Nothing.

Description

Method; cleans up after itself by freeing assets used by this component on this client and by disconnecting the component from the server.

FCRoomList.createRoom()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
roomList_mc.createRoom()
```

Parameters

None.

Returns

Nothing.

Description

Method; opens a Create Room dialog box and temporarily disables the Join, Create, and Delete buttons.

FCRoomList.deleteRoom()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
roomList_mc.deleteRoom()
```

Parameters

None.

Returns

Nothing.

Description

Method; calls the server that is requesting to delete the currently selected room.

FCRoomList.joinRoom()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
roomList_mc.joinRoom()
```

Parameters

None.

Returns

Nothing.

Description

Method; launches a new browser window to open an instance of the selected room. The application instance and user name are sent as parameters to the new window.

SetBandwidth component

This component lets users specify their upload and download bandwidths. These new settings then adjust the quality for the microphones and cameras that the component manages. It can handle multiple microphones and cameras and prioritizes bandwidth allocation.

The user has the following choices for upload and download: Modem, DSL, LAN, and Custom. When the user selects Custom, a dialog box lets the user select a fixed bandwidth number or enter a number. The bandwidths can be set to be either symmetric or asymmetric. If the user chooses the Asymmetric option, the upload and download numbers are not equal.

The SetBandwidth component uses its server-side counterpart to limit the upload and download bandwidth that the user selects. Limiting the bandwidth notifies the Flash Media Server about the available bandwidth on the connection, which generally improves the audio and video streaming quality.

You should publish at *optimal settings*—that is, at a rate no higher than what your subscribers can receive. For, example, if an online presenter has a high-speed connection, but the viewers are watching on modem and DSL connections, then the presenter should not publish any higher than modem speed. Because modem speed results in a significant reduction in quality, the presenter might want to publish at a DSL setting and indicate in the user interface that modem users' motion quality might suffer.

The SetBandwidth component does not require scripting if you use it with the SimpleConnect component.

Using the SetBandwidth component

This simple, powerful component automatically adjusts the quality of microphones and cameras that are published commensurate with the available upload bandwidth. You can use this component in all applications that publish live audio and video.

The following list describes two ways to use this component:

- Video conference: FCVideoConference uses the AVPresence component, which is designed to work with FCSetBandwidth, so the microphone and camera quality are automatically adjusted.
- Presentation: The concept of presentation applications have one presenter and many listeners. In such applications, the AVPresence component frequently provides a video presentation. It is beneficial to use with the SetBandwidth component in these applications.

To use the component in your application:

1. Make sure Flash Media Server is running.
2. Create a directory in the Flash Media Server applications directory, and name it setBand_test.
3. Create a file named main.asc in the setBand_test directory with the following code:

```
load("components.asc");
```

> **NOTE** You can also copy the main.asc file from another application directory to this application's directory.

4. In Flash, drag the SetBandwidth component onto the Stage.
5. In the Property inspector give the component an instance name, such as `setBand_mc`. To set other parameters, see "Property summary for the FCSetBandwidth class" on page 65.
6. Drag the SimpleConnect component onto the Stage.
7. In the Property inspector for the SimpleConnect component, provide the following two parameters:
   - In the Application Directory text box, type the URI of the application that you created in step 2: `rtmp:/setBand_test`.
   - Double-click the Components text box. In the Values dialog box that appears, click the plus sign (+), type the instance name of the SetBandwidth component (for example, `setBand_mc)`, and click OK.

   By providing this value, the SimpleConnect component handles the connection for the SetBandwidth component. When the SimpleConnect component successfully connects to the server, the SetBandwidth component automatically connects to the server.
8. Save and publish the file as setBand_test.swf.
9. Open the SWF file or, in the Flash authoring environment, select Control > Test Movie, and log in.

   You can set different bandwidths.

Reskinning the SetBandwidth component

The assets for this component are located in the Core Assets - Developer Only\SetBandwidth Assets directory in the Library panel.

Related components

SimpleConnect component

FCSetBandwidth class

The FCSetBandwidth class manages the quality of cameras and microphones by setting up a global Quality Manager object (`gFlashCom.quality`) that has `manage()` and `unmanage()` methods.

The following example shows two sets of microphones and cameras being registered. The first and second parameters name the microphone and camera, respectively, that are being registered. The third parameter, which is optional, sets the aspect ratio (height/width) for the camera. The last parameter determines how bandwidth is shared. In the example, mic1/cam1 gets one-third and mic2/cam2 gets two-thirds. If the last parameter is unspecified, bandwidth is equally distributed.

```
gFlashCom.quality.manage(mic1, cam1, 1, 100);
gFlashCom.quality.manage(mic2, cam2, .75, 200);
```

Unmanaging the microphones and cameras stops the Quality Manager from automatically adjusting their quality, as shown in the following example:

```
gFlashCom.quality.unmanage(mic1, cam1);
```

If your application uses the SetBandwidth component but does not use the AVPresence component, you must use the Quality Manager object to register your microphones and cameras. The AVPresence component automatically calls the Quality Manager object.

Method summary for the FCSetBandwidth class

Method	Description
`FCSetBandwidth.connect()`	Sets the bandwidth limits of the specified NetConnection object according to the currently selected bandwidths.
`FCSetBandwidth.close()`	Cleans up after itself.

Property summary for the FCSetBandwidth class

Property	Description
`FCSetBandwidth.modemUp`	Determines the upload bandwidth when you select the Modem option. The defaults are 33 kilobits per second (Kbps) up and 33 Kbps down.
`FCSetBandwidth.modemDown`	Determines the download bandwidth when you select the Modem option. The default is 33 Kbps down.

Property	Description
FCSetBandwidth.dslUp	Determines the upload bandwidth when you select the DSL option. The default is 128 Kbps up.
FCSetBandwidth.dslDown	Determines the download bandwidth when you select the DSL option. The default is 256 Kbps down.
FCSetBandwidth.lanUp	Determines the upload bandwidth when you select the LAN option. The default is 1000 Kbps up.
FCSetBandwidth.lanDown	Determines the download bandwidth when you select the LAN option. The default is 1000 Kbps down.

FCSetBandwidth.modemUp

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
setBand_mc.modemUp = maxSpeed
```

Description

Property; determines the upload bandwidth when you select the Modem option. This is a read/write property. The value *maxSpeed* is the upload speed in Kbps. The default is 33 Kbps.

FCSetBandwidth.modemDown

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
setBand_mc.modemDown = maxSpeed
```

Description

Property; determines the download bandwidth when you select the Modem option. This is a read/write property. The value *maxSpeed* is the download speed in Kbps. The default is 33 Kbps.

FCSetBandwidth.dslUp

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
setBand_mc.dslUp = maxSpeed
```

Description

Property; determines the upload bandwidth when you select the DSL option. This is a read/write property. The value *maxSpeed* is the upload speed in Kbps. The default is 128 Kbps.

FCSetBandwidth.dslDown

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
setBand_mc.dslDown = maxSpeed
```

Description

Property; determines the download bandwidth when you select the DSL option. This is a read/write property. The value *maxSpeed* is the download speed in Kbps. The default is 256 Kbps.

FCSetBandwidth.lanUp

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
setBand_mc.lanUp = maxSpeed
```

Description

Property; determines the upload bandwidth when you select the LAN option. This is a read/write property. The value *maxSpeed* is the upload speed in Kbps. The default is 1000 Kbps.

FCSetBandwidth.lanDown

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
setBand_mc.lanDown = maxSpeed
```

Description

Property; determines the download bandwidth when you select the LAN option. This is a read/write property. The value *maxSpeed* is the download speed in Kbps. The default is 1000 Kbps.

FCSetBandwidth.connect()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
setBand_mc.connect(nc)
```

Parameters

nc The active NetConnection object.

Returns

Nothing.

Description

Method; sets the bandwidth limits of the *nc* parameter according to the currently selected bandwidths.

FCSetBandwidth.close()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
setBand_mc.close()
```

Parameters

None.

Returns

Nothing.

Description

Method; cleans up after itself by freeing up assets that this component uses on this client.

Using your component with the SetBandwidth component

Once a camera and microphone are registered with the Quality Manager object (see FCSetBandwidth class), your component can subscribe to events that notify the server when the quality of a camera or microphone has changed. To listen to these events, do the following:

1. Provide an `onChanged` method, such as the one shown here:

```
listener.onChanged = function(microphone, camera)
{
  // Re-adjust the settings, etc.
}
```

2. Register your listener with the Quality Manager, as shown here:

```
gFlashCom.quality.addListener(listener);
```

These steps ensure that the `onChanged` method is called whenever the SetBandwidth component adjusts the settings of a microphone or camera.

SimpleConnect component

This component handles connecting a user to Flash Media Server and connecting any other specified components in the application to the server. The component also displays the user name and has an interface for the user to change his or her name.

The SimpleConnect component logs a user in by retrieving data from a local shared object that is created the first time the user attempts to log in. The use of the local shared object provides some continuity among connections and prevents the user from logging out of an application without exiting.

To use this component, drag it into your Flash application and edit the component parameters, which are located in the Property inspector. You need to supply the application's directory, such as rtmp:/test, and a list of any other media components that you want to connect automatically.

This component is designed to work with all other media components to simplify handling the NetConnection object. It lets you construct a new media application using media components with little or no ActionScript.

TIP	The component you want to register with the SimpleConnect component must be on the stage when the SimpleConnect component initializes. If you do not want the component to be visible in the same frame, have the component on stage but out of sight (_visible = false) in the same frame as the SimpleConnect component. Or, when the component that you want connected does appear on the stage, add an explicit component.connect(); to your code at that time.

Using the SimpleConnect component

The following list describes two ways to use this component:

- Video conference: Use this component with the VideoConference component to easily construct a fully functional application.
- Custom media components: Build media components that implement the `connect()`, `close()`, and `setUsername()` methods. You can use the SimpleConnect component to manage the network connection.

For uses of SimpleConnect, see any code example in this document.

Property inspector parameters

Name	Description
Application Directory	String; component parameter that specifies the location of the registered application's directory to use when connecting to the server.
Media Components	Array; component parameter that contains a list of other media components to be connected by FCSimpleConnect.

Reskinning the SimpleConnect component

The assets for the SimpleConnect component are located in the Core Assets - Developer Only\SimpleConnect Assets directory in the Library panel.

UserColor component

This component lets users change their selected colors for components within Flash Media Server applications. When users select a color from the menu, other components that show color, such as Chat components, change color accordingly.

The UserColor component stores the selected color in a local shared object on the user's computer. If no local shared object is found (on the first visit), a random color is selected from the list to display colors.

The UserColor component does not require scripting if you use it with the SimpleConnect component.

Using the UserColor component

This simple, visually powerful component is a centralized tool that adds color to other components.

The following list describes two ways to use this component:

- More legible chat: If you use the Chat component without the UserColor component, the text color is black for all users. Adding the UserColor component makes the chat text more legible and gives users a level of customization.
- Whenever you need virtual identity: A user-specific color is similar to a user name. The UserColor component lets you give users this enhanced sense of identity without much coding effort.

Try using the SimpleConnect, UserColor, Cursor, and Chat components together. Users can see their text and cursors showing their name, color, and position on the Stage. The user color provides for a richer user experience.

To use this component in your application, see PeopleList component.

Reskinning the UserColor component

If you want to change the colors of this component, double-click the UserColor symbol in the Library panel, unlock the combo box layer and select the ColorCombo box, open the Property inspector, and change the values in the Data array. The color palettes are updated automatically. Make sure you maintain the same number of elements in the Labels array and the Data array.

Related components

PeopleList component

FCUserColor class

The FCUserColor class provides the client-side and server-side functionality for connecting your objects to the server and cleaning up.

Method summary for the FCUserColor class

Method	Description
`FCUserColor.connect()`	Looks for a saved FCUserColor local shared object to retrieve the user's last color.
`FCUserColor.close()`	Cleans up after itself.

FCUserColor.connect()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
userColor_mc.connect(nc)
```

Parameters

nc The active NetConnection object.

Returns

Nothing.

Description

Method; looks for a saved FCUserColor local shared object to retrieve the user's last color. If the local shared object exists, the color is selected; otherwise, a random color is selected for the user and is saved in the local shared object.

FCUserColor.close()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

userColor_mc.close()

Parameters

None.

Returns

Nothing.

Description

Method; cleans up after itself by freeing assets that this component uses on this client.

Using your component with the UserColor component

If you write a component and want it to work with the UserColor component, make sure to take the following steps to monitor color change:

1. Provide an `onColorChange` method, such as the one shown here:

   ```
   MyComponentClass.prototype.onColorChange = function()
   {
     this.setColor(gFlashCom.userprefs.color);
   }
   ```

   The `gFlashCom.userprefs` global object gets updated by the FCUserColor component with the new color.

   The `this.setColor()` method of the component deals with the change in color. It's good practice to split the logic between the `onColorChange` and `setColor()` methods so you can use `setColor()` even if you don't use the UserColor component, which triggers `onColorChange()`.

2. Add your component class to the list of listeners for changes to the `userprefs` object (that includes color). After the `#endinitclip` statement of your component class, type the following code:

   ```
   gFlashCom.userprefs.addListener(this);
   ```

These steps ensure that your `onColorChange` method is called whenever you use the FCUserColor component to change color.

VideoConference component

This component lets multiple users interact with each other using audio and video, by means of the AVPresence component.

The VideoConference component does not require scripting if you use it with the SimpleConnect component.

Using the VideoConference component

You can use this versatile media component in applications to create a video conference space for users.

Drag the VideoConference, SimpleConnect, SetBandwidth, and ConnectionLight components onto the Stage to create a video conferencing application.

To use the component in your application:

1. Make sure Flash Media Server is running.
2. Create a directory in the Flash Media Server applications directory, and name it video_test.
3. In Flash, drag the VideoConference component onto the Stage.
4. In the Property inspector, give the component an instance name, such as `videoconf_mc`. To set additional parameters, see "VideoConference component Property inspector" on page 75.
5. Drag the SimpleConnect component onto the Stage.
6. In the Property inspector for the SimpleConnect component, provide the following two parameters:
   - In the Application Directory text box, type the URI of the application that you created in step 2: `rtmp:/video_test`.
   - Double-click the Components text box. In the Values dialog box that appears, click the plus sign (+) and type the instance name of the VideoConference component: `videoconf_mc`. Click OK.

   By providing this value, you are using the SimpleConnect component to handle the connection for the VideoConference component. When the SimpleConnect component successfully connects to the server, the VideoConference component automatically connects to the server.

7. Save and publish the file as video_test.swf.
8. In this application's directory, create a file named main.asc with the following code:

```
load("components.asc");
```

> **NOTE** You can also copy the main.asc file from another application directory to this application's directory.

9. Open the SWF file. (Alternatively, select Control > Test Movie in the Flash authoring environment.) Log in and click the Auto check box.

   If you have not previously set privacy settings for Flash Player, you are asked for permission to allow camera and audio access.

   If your audio device is running, the audio level moves up when you speak. If your video device is running, you can see the output in the video display. Move your mouse pointer over the video display to see the user interface for the microphone and camera as well as a button that lets you stop transmitting audio and video data.

   If the `dragSharing` property is set to `true`, you can move the video screen around.

VideoConference component Property inspector

You can set the following variables from the Property inspector for your component instance.

Name	Variable	Description
Show Boundary	`showBoundary`	A Boolean value; the default value is `false`. Determines if the boundary around the video conference area appears.
Show Background	`showBackground`	A Boolean value; the default value is `false`. Determines whether the video conference has a background or is transparent.
Clip Mask	`clipMask`	A Boolean value; the default value is `false`. Determines whether the video windows are clipped to the video conference area.
Drag Sharing	`dragSharing`	A Boolean value; the default value is `true`. Determines whether the video windows' positions are shared. Sharing lets video windows move when a remote user drags them; any drags are visible to other users.

Reskinning the VideoConference component

The assets for this component are in the Core Assets - Developer Only\VideoConference Assets directory in the Library panel.

Related components

SimpleConnect component

FCVideoConference class

The FCVideoConference class provides the client-side and server-side functionality for connecting your objects to the server and cleaning up.

Method summary for the FCVideoConference class

Method	Description
`FCVideoConference.connect()`	Initiates the video conference.
`FCVideoConference.setUsername()`	If the *`newName`* parameter is not `null` or `undefined`, the method changes the user identity; otherwise, it stops publishing the camera and microphone.
`FCVideoConference.close()`	Stops participating in the video conference.

FCVideoConference.connect()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
myVideoConf_mc.connect(nc)
```

Parameters

`nc` The active NetConnection object.

Returns

Nothing.

Description

Method; initiates the video conference. If the user name is set, this method publishes the camera and microphone.

FCVideoConference.setUsername()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
myVideoConf_mc.setUsername([newName])
```

Parameters

newName A string.

Returns

Nothing.

Description

Method; if *newName* is not `null` or `undefined`, the method changes the user identity. If *newName* is `null` or `undefined`, the publishing camera and microphone stream is closed.

FCVideoConference.close()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
myVideoConf_mc.close()
```

Parameters

None.

Returns

Nothing.

Description

Method; cleans up after itself by freeing assets used by this component on this client and by ending this client's participation in the video conference.

VideoPlayback component

The VideoPlayback component lets you control the playback of a buffered audio-video stream. You can pause or play the Flash content, jump to any position by dragging the playhead, and adjust the audio level setting.

The VideoPlayback component does not require scripting if you use it with the SimpleConnect component. The VideoPlayback component loads the stream you specified in the `streamName` variable, which you can then play by means of the `playStream()` method.

Using the VideoPlayback component

You can use this component as a video playback tool for recorded streams. The following list describes two ways to use this component:

- Flash content playback: Play back a collection of previously recorded content.
- Video notes: Play messages left earlier by another user.

To use this component in your application, see "Using the VideoRecord component" on page 80.

VideoPlayback component Property inspector

You can set the following variables from the Property inspector for your component instance.

Name	Variable	Description
Default Stream Name	`streamName`	String; the default value is `video`. Specifies the name of the stream to initially load and play back.
Buffer Time	`bufferTime`	Number; the default value is 2. Specifies the length, in seconds, of the video data to buffer for the playback.

Reskinning the VideoPlayback component

The assets for this component are in the Core Assets - Developer Only\VideoPlayback Assets directory in the Library panel.

Related components

SimpleConnect component and VideoRecord component

FCVideoPlayback class

The VideoPlayback class provides the client-side and server-side functionality for connecting your objects to the server and cleaning up.

Method summary for the FCVideoPlayback class

Method	Description
`FCVideoPlayback.connect()`	Sets up the assets required by the VideoPlayback component (on the client side and the server side).
`FCVideoPlayback.close()`	Cleans up and shows the normal mouse pointer (`Mouse.show`).

FCVideoPlayback.connect()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
vidPlayback_mc.connect(nc)
```

Parameters

nc The active NetConnection object.

Returns

Nothing.

Description

Method; sets up all the assets needed by the VideoPlayback component (on the client side and the server side).

FCVideoPlayback.close()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
vidPlayback_mc.close()
```

Parameters

None.

Returns

Nothing.

Description

Method; closes the component and cleans up by freeing assets used by this component on this client.

VideoRecord component

The VideoRecord component records a buffered stream to the server. The camera and microphone output is recorded.

Using the VideoRecord component

You can use this component as a video recorder, saving streams to the server. The following list describes two ways to use this component:

- Movie recorder: Use this component to capture video to the server for later playback.
- Video notes: Record messages for another user to view later.

To use the component in your application:

1. Make sure Flash Media Server is running.
2. Create a directory in the Flash Media Server applications directory, and name it recplay_test.
3. Drag the VideoRecord component onto the Stage and give it an instance name, such as `record_mc`. To set additional parameters, see "VideoRecord component Property inspector" on page 82.
4. Drag the VideoPlayback component onto the Stage and give it an instance name, such as `playback_mc`. To set additional parameters, see the "VideoPlayback component Property inspector" on page 78.
5. Drag the SimpleConnect component onto the Stage.

6. In the Property inspector for the SimpleConnect component, provide the following two parameters:
   - In the Application Directory text box, type the URI of the application you created in step 2: `rtmp:/recplay_test`.
   - Double-click the Components text box. In the Values dialog box that appears, click the plus sign (+) and type the instance name of the playback and record components: `playback_mc` and `record_mc`. Click OK.

   By providing these values, you are using the SimpleConnect component to handle the connection for the VideoPlayback and VideoRecord components. When the SimpleConnect component successfully connects to the server, the other components automatically connect to the server.
7. Save and publish this file as recplay_test.swf.
8. In the application's directory, create a file named main.asc with the following code:

```
load("components.asc");
```

NOTE	You can also copy the main.asc file from another application directory to this application's directory.

9. Open the SWF file or, in the Flash authoring environment, select Control > Test Movie, and log in.

   Click the Record button of the VideoRecord component to record the default stream `video` as defined in the component's Default Stream Name property. The camera and microphone output is recorded when the right-arrow Record button is clicked on the VideoRecord component.

   The stream to play back, `video`, is defined in the first property of the playback component. After you have stopped recording, click the right-arrow Playback button on the playback component to see the recorded video stream play back.

VideoRecord component Property inspector

You can set the following variables from the Property inspector for your component instance.

Name	Variable	Description
Default Stream Name	`streamName`	String; the default value is `video`. Specifies the name of the stream to initially load and play back.
Default Settings	`[setLow setMed setHigh]`	String; the default value is `setHigh`. Indicates the camera settings to use when the component is initialized. The three possible settings are `setLow`, `setMed`, and `setHigh`.
Buffer Time	`bufferTime`	Number; the default value is 10. Indicates the length, in seconds, of the video data to buffer for recording.
Low Quality Setting	`setLow`	Object; contains the following values to use for the camera settings. width, 160 horizontal pixels height, 120 vertical pixels fps, 10 video frames per second bandwidth, maximum bandwidth to use, 0 (0 indicates unlimited) quality, video compression quality, 50 (0 indicates unlimited) key, keyframe interval, 10 keyframes rate, audio quality in Khz, 11 Khz

Name	Variable	Description
Medium Quality Setting	`setMed`	Object; contains the following values to use for the camera settings. width, 240 horizontal pixels height, 180 vertical pixels fps, 12 video frames per second bandwidth, maximum bandwidth to use, 0 (0 indicates unlimited) quality, video compression quality, 80 (0 indicates unlimited) key, keyframe interval, 12 keyframes rate, audio quality in Khz, 22 Khz
High Quality Setting	`setHigh`	Object; contains the following values to use for the camera settings. width, 320 horizontal pixels height, 240 vertical pixels fps, 15 video frames per second bandwidth, maximum bandwidth to use, 0 (0 indicates unlimited) quality, video compression quality, 90 (0 indicates unlimited) key, keyframe interval, 5 keyframes rate, audio quality in Khz, 44 Khz

Reskinning the VideoRecord component

The assets for this component are in the Core Assets - Developer Only\VideoRecord Assets directory in the Library panel.

Related components

SimpleConnect component and VideoPlayback component

FCVideoRecord class

The VideoRecord component does not require scripting if you use it with the SimpleConnect component. If you do not set the `streamName` property, the component records to the default stream name.

Method summary for the FCVideoRecord class

Method	Description
`FCVideoRecord.connect()`	Sets up the assets required by the VideoRecord component (on the client side and the server side).
`FCVideoRecord.close()`	Cleans up after the component and shows the normal mouse pointer (`Mouse.show`).

FCVideoRecord.connect()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
vidRecord_mc.connect(nc)
```

Parameters

nc The active NetConnection object.

Returns

Nothing.

Description

Method; sets up all the assets needed by the VideoRecord component (on the client side and the server side).

FCVideoRecord.close()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
vidRecord_mc.close()
```

Parameters

None.

Returns

Nothing.

Description

Method; closes the component and cleans up by freeing assets used by this component on this client.

Whiteboard component

This component lets you create and edit text, boxes, and lines in a real-time shared environment. To create a shape, you select a tool and click on the whiteboard area; you can also drag the shape to a new position or edit the text. When you press the Delete key, all actively selected items are deleted.

The Whiteboard component does not require scripting if you use it with the SimpleConnect component.

Using the Whiteboard component

You can use this component to create a collaborative environment for developing ideas. The following list describes several ways to use this component:

- Shared whiteboard: Map out ideas with other users.
- Organization chart: Outline the company's organizational structure.
- Workflow: Display the ongoing progress of a project. As additional phases are completed, they can be updated on the whiteboard so all users can see the current status.

To use the component in your application:

1. Make sure Flash Media Server is running.
2. Create a directory in the Flash Media Server applications directory, and name it whiteboard_test.
3. Drag the Whiteboard component onto the Stage.
4. In the Property inspector, give the component an instance name, such as `whiteboard_mc`.
5. Drag the SimpleConnect component onto the Stage.

6. In the Property inspector for the SimpleConnect component, provide the following two parameters:
   - In the Application Directory text box, type the URI of the application that you created in step 2: `rtmp:/whiteboard_test`.
   - Double-click the Components text box. In the Values dialog box that appears, click the plus sign (+) and type the instance name of the Whiteboard component: `whiteboard_mc`. Click OK.
7. Save and publish the file as whiteboard_test.fla.
8. In this application's directory, create a file named main.asc with the following code:

```
load("components.asc");
```

NOTE	You can also copy the main.asc file from another application directory to this application's directory.

9. Open two instances of the SWF file or, in the Flash authoring environment, select Control > Test Movie, and log in.

   You see the whiteboard with several tools on the left. If you open more instances of the application, each instance displays the shapes and text of the other instances. You can select and move objects, type text, create text boxes, draw arrows, and select colors.

Reskinning the Whiteboard component

The assets for this component are in the Core Assets - Developer Only\Whiteboard Assets directory in the Library panel.

Related components

SimpleConnect component

FCWhiteboard class

The FCWhiteboard class provides the client-side and server-side functionality for connecting your objects to the server and cleaning up.

Method summary for the FCWhiteboard class

Method	Description
`FCWhiteboard.connect()`	Sets up all the assets needed by the component (on the client side and the server side).
`FCWhiteboard.close()`	Cleans up after itself by freeing assets used by this component on this client.

FCWhiteboard.connect()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

wb_mc`.connect(`*nc*`)`

Parameters

nc The active NetConnection object.

Returns

Nothing.

Description

Method; sets up all the assets needed by the component (on the client side and the server side).

FCWhiteboard.close()

Availability

- Flash Player 6.
- Flash Communication Server MX.

Usage

```
wb_mc.close()
```

Parameters

None.

Returns

Nothing.

Description

Method; cleans up after itself by freeing assets used by this component on this client.