

Exercise 1.1 – Creating your own Error Handling for Flash Objects

Edward J. Apostol, consultant

New Toronto Group

Thanks to Derrick Ypenburg who originally created this exercise

1. Open the file **loadXML_start fla**.
2. In the timeline, insert a new layer that will temporarily create our error panel.
3. Draw a simple rectangle with a colored background of your choice.
4. Convert the rectangle object to a movie clip symbol called **ErrorPanel** (Select the object and press F8)
5. Double-click directly on your newly created **ErrorPanel** to edit it in place.
6. Select Layer 1 in the ErrorPanel movie clip's timeline and rename it to **Background**.
7. Still in the ErrorPanel's timeline, create a new layer named **Display Message**.
8. Draw a Dynamic text field of whatever size, font, and color you'd like to use.
9. Give the text field an instance name of **tDisplayMessage**. Make sure the Selectable option in the text field Property inspector is de-selected.
10. Create a new layer named **Button** and create a button symbol of your liking. Give the button symbol an instance name of **btnOK**.
11. Create a new layer named **Actions** and add the following ActionScript to the layer:

```
var sErrorType:String;

function errorMessage():Void {
    switch(sErrorType) {
        case "xmlLoadFail":
            this.tDisplayMessage.text = "XML has failed to load. Please try again later.";
            break;
    };
};

btnOK.onRelease = function():Void {
```

```
        this._parent.unloadMovie();
    };

    errorMessage();
```

The code first declares a new variable called **sErrorType**. This will be used to store a message that will be displayed in the text field. The value will get passed in an object from the main timeline when the **ErrorPanel** is attached dynamically.

Next, a function named **errorMessage** is created. This will use a switch statement to match the type of error that was passed to the **sErrorType** string and display a message accordingly. You can categorize as many different error types and actions here as you'd like by adding different case statements.

An event handler is then created for **btnOK** that unloads the **ErrorPanel** when pressed.

The **errorMessage** function is called when the **ErrorPanel** is attached to the stage and the movieclip run its code.

12. Right-click (PC) or Control-click (Mac) **ErrorPanel** in the Library and select **Linkage** from the context menu. The **Linkage** panel will appear.
13. Check the **Export for ActionScript** option. The same name (with no spaces) as the symbol's Library name will automatically show up in the Identifier field (**ErrorPanel**).
14. Click **OK**. We will use this identifier name to attach the **ErrorPanel** from the Library when an error occurs.
15. Return to the main timeline.
16. Select the **Actions** layer of the main timeline and insert the following code just below the **onLoad** block of code:

```
// Function called when error occurs
function launchError(sError:String):Void {

    this.attachMovie("ErrorPanel", "mcErrorPanel", 1, {_x:11, _y:10, sErrorType:sError});

};
```

When the function is called, a string value gets passed in that describes the type of error that has occurred (**sError**).

The function then uses the attachMovie method of ErrorPanel to place the symbol from the library to the stage, gives it an instance name of mcErrorPanel and places it on a higher depth than the main timeline.

The fourth parameter in the attachMovie method passes an object to mcErrorPanel. It passes an _x and _y position for the instance and also passes the value of sError to the string variable sErrorType inside of the movie clip's timeline (remember, this is the variable that the switch statement references to determine what error has happened and what to do).

17. The last thing we need to do here is call the launchError function when an error occurs. Let's do this from within the else statement of the xmlLoad.onLoad event block of code:

```
xmlLoad.onLoad = function(success:Boolean):Void {
    if (success) {
        // XML file has successfully loaded
        //use data that is loaded
        tMessage.text = this.firstChild.childNodes[0].firstChild;
    } else {
        // XML file could not be loaded
        // inform user that data could not load and handle error internally

        // Call error function and pass error type
        launchError("xmlLoadFail");

        // Set text field
        tMessage.text = "XML Data could not load. Please try again later.";
    };
};
```

Exercise 1.2 – Working with Messages received from Flash Video-Related Objects

Edward J. Apostol, consultant

New Toronto Group

Thanks to Derrick Ypenburg who originally created this exercise

Objectives:

In this exercise you will build the following:

- A simple Flash-based video player
- A connection to an FMS Server
- Code that will handle messages received while video is streaming



Basic Setup

1. Open the **videoPlayer_start fla**.
2. View the file and note that there is an instance of a MovieClip called mcVideoPlayer on the main timeline.
3. Edit mcVideoPlayer in place (double-click the instance), and note what's inside of mcVideoPlayer's timeline. On a layer named **Controls**, there are three MovieClips for our player controls (mcPlayButton, mcPauseButton and mcStopButton; the buttons are already set-up and working in their timelines). We will write the code for these to control our video player.

4. On the **Video** layer, there is an embedded video object called videoIn. We will display the video playing through the NetStream instance we will create in this object.
5. On the **Status Text** layer is a graphic and a text field named tStatus. We will display values in this text field informing the user of the current playback state of the video.
6. There is a layer named **Actions** which we will add to shortly. It currently has a few lines of code that declare the objects we'll be controlling, as variables.
7. Return to the main timeline.
8. Note in the library there is a MovieClip named **ErrorPanel**. This is the same ErrorPanel MovieClip that was created in the previous exercise. We will attach the MovieClip to the stage when the launchError function on the **Actions** layer gets called from the mcVideoPlayer instance if an error occurs (we will build this functionality in a little bit).

This is our basic set-up for the player. Next, we will start adding the ActionScript to manage our application.

Setting Up the NetConnection Class

We'll start off by instantiating a NetConnection object (it has already been declared as a variable at the top of the script). We'll then call the connect method of this instance, nc, and pass the path to the a local video application instance (a folder) on the Flash Media Server, as a parameter.

9. Add the following code after the variable declaration code on the Actions layer, and uncomment the appropriate line of code that you will use to connect to the FMS Server

```
// Instantiate NetConnection
var nc:NetConnection = new NetConnection();

// connect nc to FMS

/* use this address if you do not have access to an FMS Server by uncommenting the line
below */
// nc.connect("rtmp://flashcom.mynetkeepers.ca/ypenburg/communitymx/");

// otherwise, use this address if you are using a local version of FMS
// nc.connect("rtmp://localhost/seaturtle/");
```

The NetConnection class also broadcasts an event named onStatus. This event is how we can determine if we had a successful or unsuccessful connection, or a break or error in the connection during the persistent connection.

If we create an event handler for this event, the NetConnection object passes an information object, ncObj, as a parameter. We can then call the object's code property and view its value to see what is going on with the connection.

10. Add the following ActionScript to assign a function (ncStatus) to the onStatus event of nc:

```
// onStatus event for nc
nc.onStatus = Delegate.create(this, ncStatus);

function ncStatus(ncObj:Object):Void {
    trace(ncObj.code);
};
```

11. Save and preview your file.

Note the ncObj.code value you see traced to the output window. If you see "NetConnection.Connect.Success", you have successfully connected to the correct applications directory on the FMS server.

If you see a message such as "NetConnection.Connect.Rejected" and/or "NetConnection.Connect.Closed", you need to check the path in the connect method of nc and make sure it is correct, or check your Internet connection — or my server is down at the time you're doing this.

***Note:** We are not checking for port and protocols that may be required to establish a NetConnection from behind some firewalls, so that can also be an issue if you have a firewall that blocks RTMP connection protocols.*

12. Once you get a successful connection, let's add some code inside of the onStatus event handler to handle both successful and unsuccessful connections.

If the connection is successful, we'll trace a message out, for now. If the connection is unsuccessful, let's call the launchError function in the main timeline and pass a "NetConnection" string to it as a parameter.

This will then be displayed as a proper error message in the error panel MovieClip when it is launched.

```
// onStatus event for nc
function ncStatus(ncObj:Object):Void {
    trace(ncObj.code);
    if (ncObj.code == "NetConnection.Connect.Success") {
        trace("You have successfully connected to the server");
    } else {
        this._parent.launchError("NetConnection");
    };
};
```

You can match all the literal values in your “if” statement to handle any of the six event object messages that are passed by the NetConnection object.

Working with the NetStream Object

Once we get a successful NetConnection, we can then instantiate a NetStream object. Some developers instantiate a NetStream at the same time as a NetConnection, but we only need a NetStream if we get a successful NetConnection for the NetStream to connect through.

Let's modify our previous code to only instantiate and connect a NetStream instance (ns) if we are successfully connected. We will also assign a function to the onStatus event. We then attach ns to the videoIn object on the stage.

13. Modify the ncStatus event handler function as follows:

```
function ncStatus(ncObj:Object):Void {
    if (ncObj.code == "NetConnection.Connect.Success") {
        // Instantiate NetStream only on successful NetConnection
        ns = new NetStream(nc);
        // Function for onStatus event of ns
        ns.onStatus = Delegate.create(this, nsStatus);
        // Attach ns to videoIn object
        videoIn.attachVideo(ns);
    } else {
        this._parent.launchError("NetConnection");
    };
};
```

14. After the nc onStatus event block of code, add the following code for the function that gets called for the ns onStatus event:

```
// function called by ns onStatus event
function nsStatus(nsObj:Object):Void {
  // Status message passed through object
  trace(nsObj.code);
};
```

Notice that the onStatus event for a NetStream instance, handled by the nsStatus() function, also passes an information object (nsObj). We can get the value of this object's code property and match up its values to get the status of the NetStream.

We won't see any status messages yet, as the onStatus event only gets called when we try and play a video through the NetStream instance. We will come back to this shortly.

Controlling the NetStream

Let's quickly wire up our control buttons to play, pause, and stop the NetStream instance (ns) so we can see the different status messages traced to the Output panel.

15. Add the following events to the mcPlayButton, mcPauseButton, and mcStopButton instances:

```
// Control event handlers
mcPlayButton.onRelease = function():Void {
  // this._parent.ns.play("cafe_townsend_chef", 0);
  // substitute video file here as appropriate
  this._parent.ns.play("seaturtle300", 0);
};

mcPauseButton.onRelease = function():Void {
  this._parent.ns.pause();
};

mcStopButton.onRelease = function():Void {
  this._parent.ns.close();
};
```

When mcPlayButton is released, the play() method of ns is called. The two parameters passed to the method are:

- "cafe_townsend_chef" / "seaturtle300" – the name of the FLV to play. The path to the video is already established in the NetConnection instance, so all we have to do is call the name of the video **WITHOUT** the .flv extension. (When calling video from a FMS, you do not need to specify the .flv extension.)
- The second parameter specifies whether the video is live or not. 0 means the video is on-demand and should be found on the server. A value of 1 would tell the NetStream instance to wait for the live stream name to start being published if it does not exist on the server at the time it was called.

When mcPauseButton is released, the pause() method of the NetStream instance is called (if a NetStream instance is playing and told to pause, it pauses; if a NetStream instance is *paused* and told to pause, it plays again [unpauses]).

When mcStopButton is pressed, the NetStream instance is closed (there is no stop() method for the **NetStream** instance).

16. Save and preview your file. Perform the following steps and take note of the NetStream event messages that get traced to the Output panel (ignoring the Buffer event messages):

- Press the play button to play the video. Note the NetStream event message that gets traced to the Output panel:

NetStream.Play.Reset
NetStream.Play.Start

- Press the pause button to pause the video. Note the NetStream event message that gets traced to the Output panel:

NetStream.Pause.Notify

- Press the pause button again to un-pause the video. Note the NetStream event message that gets traced to the Output panel:

NetStream.Unpause.Notify
NetStream.Play.Start

- Press the stop button to close the **NetStream**. Note the event message that gets traced to the output panel:

NetStream.Play.Stop

We will use these event messages to:

- Display a text status message of the state of the NetStream instance
- Use the state of the NetStream instance to toggle the controls of the player

Adding our Status Messaging

Informing the user of the state of a video stream is very important, as video can be unpredictable on the Internet. Our DVD players tell us when a video is playing, paused, and stopped, so why not have our Flash video player do the same thing? Beyond that, we also want to inform the user if a video cannot be found.

17. Add the following code within the nsStatus event handler

```
// function called by ns onStatus event
function nsStatus(nsObj:Object):Void {
    switch (nsObj.code) {
        case "NetStream.Play.Start":
            videoPlaying();
            break;
        case "NetStream.Pause.Notify":
            videoPaused();
            break;
        case "NetStream.Play.Stop":
            videoStopped()
            break;
        case "NetStream.Play.StreamNotFound":
            this._parent.launchError("NetStream");
            break;
    }
};
```

The code above calls different functions depending on the NetStream object event message that gets matched up in the switch statement. When a message matches up in a case, the associated function is fired off. In the case of the StreamNotFound message, the launchError function in the main timeline gets called and the type of error ("NetStream") is passed to the function. (The NetStream error case has already been coded in the **ErrorPanel** MovieClip).

Lets write the functions that will be called by the case statements in the nsStatus function. The functions will set the text properties of tStatus depending on the state of the NetStream instance.

18. Add the following code below the nsStatus function:

```
function videoStopped():Void {
    tStatus.text = "Video is stopped";
};

function videoPaused():Void {
    tStatus.text = "Video is paused ...";
};

function videoPlaying():Void {
    tStatus.text = "Video is playing ...";
};
```

We now have status text informing the user of when the video is playing, paused, stopped, and when a video cannot be found.

Given that we have these functions in place, we can add some code to these functions that will enable and disable the player controls when they are not needed during different states of playback.

Toggling Controls

We can also use the same switch statement to toggle / disable the controls of the player when they are not needed. If the video is playing, the user should be able to click the play button again (that will re-start the video from the beginning). If the video is stopped, the stop and pause buttons should be disabled, as they are not needed, and so on.

19. Add the following code to enable and disable the player controls during playback:

```
function videoStopped():Void {
    tStatus.text = "Video is stopped";
    mcPlayButton.enabled = true;
    mcPlayButton._alpha = 100;
    mcPauseButton.enabled = false;
    mcPauseButton._alpha = 50;
    mcStopButton.enabled = false;
    mcStopButton._alpha = 50;
};

function videoPaused():Void {
    tStatus.text = "Video is paused ...";
    mcPlayButton.enabled = false;
    mcPlayButton._alpha = 50;
    mcPauseButton.enabled = true;
};
```

```
mcPauseButton._alpha = 100;  
mcStopButton.enabled = true;  
mcStopButton._alpha = 100;  
};  
  
function videoPlaying():Void {  
    tStatus.text = "Video is playing ...";  
    mcPlayButton.enabled = false;  
    mcPlayButton._alpha = 50;  
    mcPauseButton.enabled = true;  
    mcPauseButton._alpha = 100;  
    mcStopButton.enabled = true;  
    mcStopButton._alpha = 100;  
};
```

20. To finish things off, call the videoStopped() function at the bottom of all the script. This will toggle the buttons to a default state which only enables the play button

```
// call video stopped function to enable play button  
videoStopped();
```

21. Save and preview your file. Notice now how the player controls are managed by the state of the NetStream object. This is a much more air-tight approach to toggling controls.

Exercise 1.3: Using Embedded Cue Points

Edward J. Apostol, consultant
New Toronto Group

In this walkthrough, you will perform the following tasks:

- Build a “Talking Head” application
- Embed cue points into an FLV
- Use cue points to manage Flash timeline

Steps

1. Open the file **cuePointsEx1_raw fla.**
2. Save it as **cuePointsEx1 fla.**

Embedding Cue Points into an FLV

3. Click the **FLVPlayback** layer.
4. Select **File > Import > Import Video...**
5. Browse to the video **JamieOnWhite.avi.**
6. Click **Next** twice to view the **Encoding** options.
7. Click **Show Advanced Settings.**
8. Click the **Cue Points** tab.
9. Drag the play head bar underneath the video preview window to approximately 1 second.
10. Click the “+” symbol at the top of the Cue Points panel to add a cue point at the selected time in the video.
11. Name the new cue point **slide1.**
12. Repeat steps 9 – 10 using the following values:
 - a. Time: **4** seconds, Cue Point name: **slide2**
 - b. Time: **8** seconds, Cue Point name: **slide3**
 - c. Time: **10** seconds, Cue Point name: **slide4**
 - d. Time: **12** seconds, Cue Point name: **slide5**
13. Click **Next** when done.

14. Select **None** from the skin drop down menu.

15. Click **Next**.

16. Click **Finish** to encode the video.

Setting-Up the Timeline

17. Click the **FLVPlayback** component on the Stage.

18. Position the **FLVPlayback** component at X: **0** and Y: **0**.

19. Set the **Instance Name** of the **FLVPlayback** component to **videoPlayback** in the Properties panel.

20. Note the Frame label names on the Frames layer.

21. Click the **Frames** layer.

22. Add a new layer and name it: **Actions**.

23. Open the Actions panel.

24. Stop the timeline and import the video class by adding the following ActionScript:

```
stop();  
import mx.video.*;
```

25. Create a Listener for a cue point event in the video and go to the corresponding frame label name of the cue point by adding the following ActionScript:

```
videoPlayback.addEventListener("cuePoint",  
onCuePoint);  
  
function onCuePoint(evntObj:Object):Void {  
    var targetFrame:String = evntObj.info.name;  
    gotoAndStop(targetFrame);  
};
```

26. Save and Preview your movie.

Note: As the video plays, the embedded cue point events should go to their corresponding frames within the timeline.

Exercise 1.4: Adding Closed Captioning to Flash Video

In this walkthrough, you will perform the following tasks:

- Import stored captions from an XML file
- Add cue points to an FLVPlayback component using ActionScript
- Populate a Text Area component with captions that is synchronized with a video

Steps

View the XML Document

1. Open Dreamweaver 8 or a text editor.
2. Open the file **captions.xml**.
3. Note the caption times and text listed in the XML nodes of the document.
4. Close Dreamweaver 8 or the text editor.

Adding the Captions

5. Open Flash Professional 8.
6. Open the file **closedCaptioningEx fla**.
7. Click the Video Playback layer.
8. Add a new layer and name it **Caption Panel**.
9. Select the **Caption Panel** layer.
10. Drag an instance of the Text Area component from the Components panel to the stage.
11. Set the following properties for the Text Area component:
Instance name: **captions_txt**
W: **240**, H: **65**, X: **175**, Y: **255**
12. Add a new layer above the Caption Panel layer and name it: **Actions**.
13. Select the **Actions** layer.

14. Launch the **Actions panel** (F9).

15. Import the video classes of the FLVPlayback component:

```
import mx.video.*;
```

16. Create an array that will store the XML captions:

```
var captions:Array;
```

17. Instance a new XML object named **captionsXML**. Set it to ignore white space within the XML document that will be loaded.

```
var captionsXML:XML = new XML();  
captionsXML.ignoreWhite = true;
```

18. Create an onLoad method of the XML class by adding the following ActionScript:

```
captionsXML.onLoad = function():Void {  
};
```

19. Create a variable named captions with the onLoad event and set it to be equal to the number of child nodes in the XML document by adding the following ActionScript:

```
captions = this.firstChild.childNodes;
```

20. Create a **for** loop and use the current length of the captions array to set how many times the for loop will loop by adding the following ActionScript within the onLoad method:

```
for(var i:Number = 0; i < captions.length; i++){  
};
```

21. Create variables **captionTime** and **captionValue** to store the caption time and the caption values stored in the index of the captions array using the current value of **i** in the **for** loop by adding the following code within the **for** loop:

```
var captionTime:Number = captions[i].attributes.start;  
var captionValue:String = captions[i].firstChild.nodeValue
```

22. Add cue points to the FLVPlayback component by adding the following ActionScript within the for loop:

```
videoPlayback.addASCuePoint(captionTime,captionValue);
```

23. Load the caption.xml file into the captionsXML object by adding the following ActionScript:

```
captionsXML.load("captions.xml");
```

24. Add a cue point listener to the videoPlayback object that will call a function named onCuePoint every time a cue point is detected. Use the onCuePoint function to set the captions_txt component to display the value of the current cue point.

```
videoPlayback.addEventListener("cuePoint",onCuePoint);  
  
function onCuePoint(evtObj:Object):Void {  
    captions_txt.text = evtObj.info.name;  
};
```

25. Save and test your movie.

Note: You should see the captioning text from the caption.xml file displaying in the text area in sync with the speaker.