

# Embedding Video

---

## Scope of the exercise

Please ensure that you have seen the video *Embedding video in your application* before going through this exercise. This exercise is different from the others for Day 5. We will NOT be creating new code in this exercise; rather we will be reviewing the code created in the video.

The starter file that you will import has all the code to run the application. We will only be delving into a portion of that code. The application uses the as3syndication library to parse the RSS feed for the video application. The library has also been extended to handle the blip TV RSS feed. We will NOT be reviewing the code for the as3syndication library. Please look at the link provided in step 8 to download and understand the source code of the library.

The focus of this exercise is to understand how to use the VideoDisplay control and integrate video in a flex application.

## Import the Project

1. If not already created, create a directory named `adobeFlexTraining` on your **C** drive.
2. In Flex Builder, select **File > Import > Flex Project**.
3. In the dialog window, select **Archive File** and browse to where `Ex25_Starter.zip` is located in your local file system.
4. Uncheck **Use default location**.
5. Browse to `C:\adobeFlexTraining\FlexVideoDemo`.
6. Click **Finish**.

## as3syndication Library

7. The **as3syndication** library is available at the following link:

<http://code.google.com/p/as3syndicationlib/>.

We can use the syndication library to parse Atom and all versions of RSS easily. This library hides the differences between the formats so you can parse any type of feed without having to know what kind of feed it is.

For the purposes of this exercise, the library is extended to handle the blip TV RSS feed, which has FLVs and integrates with Flex easily.

**Note:** FLV stands for **Flash Video** and is the name of a file format used to deliver video over the Internet using Adobe Flash Player.

**Note:** RSS is a family of Web feed formats used to publish frequently updated works—such as blog entries, news headlines, audio, and video—in a standardized format. An RSS document (which is called a "feed", "web feed", or "channel") includes full or summarized text, plus metadata such as publishing dates and authorship.

## Integrating video from a RSS feed

- Using the **Flex Navigator** view, open the `FlexVideoDemo.mxml` located in the **FlexVideoDemo > src** folder.
- The variable declaration for `flvRssURL` points the URL link to the location of the RSS feed.

```
private var flvRssURL:String =  
    "http://wildcaster.blip.tv/rss";
```

- Within the main `Application` tag, notice the call to the function `applicationComplete()`.

```
creationComplete="applicationComplete()"
```

- Locate the `applicationComplete()` function and notice how the RSS feed is being loaded into the application. The variable `request` in the `applicationComplete()` function points to the feed that is being loaded into the application.

```
var request:URLRequest = new URLRequest(flvRssURL);  
request.method = URLRequestMethod.GET;
```

- The following code shows how to initialize the `URLLoader` object, assign an `EventListener`, and load the specified `URLRequest` object. The function `onDataLoad()` gets called when the URL gets loaded.

```
loader = new URLLoader();  
loader.addEventListener(Event.COMPLETE, onDataLoad);
```

```
loader.load(request);
```

13. Within the function `onDataLoad()`, we get the RSS data in a string format and pass it to another function `parseRSS()`.

```
private function onDataLoad(e:Event):void
{
    var rawRSS:String = URLLoader(e.target).data;
    parseRSS(rawRSS);
}
```

14. Within the function `parseRSS()`, we use the `as3syndication` library to parse the raw RSS data, and filter out anything that is not a FLV. The code below shows how the data passed to the function is parsed and stored in an `ArrayCollection`. We then loop over all the items in the array and remove any item which is not a FLV. The `loadVideo()` function is then called to load the actual video into the player.

```
private function parseRSS(data:String):void
{
    var rss:BlipTVRSS = new BlipTVRSS();
    rss.parse(data);

    items = new ArrayCollection(rss.items);

    //filter out the items that do not have
    // flvs
    for each(var item:BlipTVItem in items)
    {
        if(!item.flv)

            items.removeItemAt(items.getItemIndex(item));
    }

    loadVideo();
    videoContainer.enabled = true;
    dispatchEvent(new Event("done"));
}
```

15. The `loadVideo()` function uses the index of the video selected by the user in the `ComboBox` to find the correct video clip in the array. If the video is being currently displayed, it is stopped. If not, the `VideoDisplay` component starts the video.

```
public function loadVideo():void
```

```

        {
            var selectedIndex:uint = chooser.selectedIndex;
            var video:BlipTVItem =
items.getItemAt(selectedIndex) as BlipTVItem;
            if(videoDisplay.playing)
                videoDisplay.stop();
            videoDisplay.visible = true;
            videoDisplay.source = video.flv.url;
        }

```

16. We can add some controls to the video display as illustrated in the code below.

The `<mx:ComboBox>` has the array as a `dataProvider` which stores all the video clips. Any change within the combo box will fire the `loadVideo()` function.

The `VideoDisplay` component has the `autoplay` property set to `false`, and calls the `handlePlayheadUpdate()` function which is explained in the next step.

There is a button which can be used to play or pause the videos and calls the `handlePlayPause()` function to achieve this.

The `HSlider` component can be used to monitor the progress of the video.

The `<mx:Label>` with an `id` property of `time` displays how much time is left in the video being played.

```

<mx:VBox id="videoContainer" enabled="false">

    <mx:ComboBox id="chooser"
        dataProvider="{items}"
        change="loadVideo()"
        labelField="title"
        width="480" />

    <mx:Canvas>

        <mx:VideoDisplay id="videoDisplay"
            autoplay="false"
            width="480" height="270"
            playheadUpdate="handlePlayheadUpdate(event)" />

    </mx:Canvas>

    <mx:HBox width="100%" >

        <mx:Button id="playPause"
            label="Play"
            click="handlePlayPause()"

```

```

        width="70" />

        <mx:HSlider id="progress"
            minimum="0" maximum="1"
            width="100%" />

        <mx:Label id="time" text="0:00/0:00"/>

    </mx:HBox>

</mx:VBox>

```

17. The function `handlePlayHeadUpdate()` handles the event dispatched by the component `VideoDisplay` when the play head changes. The value for the `text` property of the `<mx:label>` `time` is computed by displaying the values of where the play head is currently in the display and the total time of the video. The value of the `HSlider` component which has an `id` of `progress` is calculated by dividing the total time of the video from where the play head is currently.

```

private function handlePlayheadUpdate(e:VideoEvent):void
{
    time.text =
Math.floor(videoDisplay.playheadTime)+"/"+Math.floor(videoDisplay
.totalTime);
    progress.value =
videoDisplay.playheadTime/videoDisplay.totalTime;
}

```

18. Using the **Flex Navigator** view, open the `Styles.css` located in the **FlexVideoDemo > src** folder. The style sheet is used to provide the African theme to the `VideoDisplay`.
19. Run the file `FlexVideoDemo.mxml` to see how the `VideoDisplay` component of Flex handles the RSS feed and displays the videos.

Note: If you experience a security sandbox error. In Flex Builder, right-click on the **FlexVideoDemo** project and select **Properties**. From the left hand menu select **Flex Compiler**. Add `-use-network=false` to your compiler arguments. Click Ok.

