



DMP 4310G Content Development Guidelines

March 29, 2011

855 Tasman Drive
MILPITAS, CALIFORNIA 95035

Table of Contents

Purpose	3
Targeted Audience	3
Prerequisites	3
DMP 4310 Supported formats	3-5

Content Examples

Videos

1.1. Develop code for playing MPG through flash	6
1.2. Creation of the video playlist.....	7
1.3. Creating output box for MediaPlayer on the fly	8
1.4. How to set correct path to .mpg files	8-9
1.5. Decreasing intervals between movies in the list	9
1.6. Playing movies from XML file list.....	9-10

XML

Loading data from static XML	11-12
------------------------------------	-------

Tickers

Typewriter ticker	12-13
Fade-out ticker	13-14

Clocks

Digital Clock	14-15
---------------------	-------

Image Slideshow

Adding alpha transition	15-16
-------------------------------	-------

RSS Feeds

What is RSS?	16-17
How to embed RSS feed into my sign?	17-18
Overcoming cross-domain issues with proxy.....	18-19

Database

Flash-Database communications	19
-------------------------------------	----

Scripting

What is ActionScript	19
Value of AS2	19-20
AS2 vs. JS	20
AS2 vs. AS3	20-23
What is Fflash Lite?	23
DMP AS vs JS API	23
Avoid Common Mistakes	24

Tuning

Tweening and Transparencies.....	24
General Rules for Flash Optimization.....	24-25
Action Script Optimization.....	25-27

DMP 4310 API

About DMP 4310 API.....	27
2.1. Moving Video Box across the screen.....	27
2.2. Changing Volume	27-28
2.3. Manipulating with Video Brightness, Contrast and Saturation.....	28
2.4. Playing video URL to coordinates	29
2.5. Playing video playback to coordinates.....	29-30
2.6. Setting and getting mib parameters of the DMP	30
2.7. Logging to Syslog on the DMP	30
2.8. Playing and stopping video on the DMP.....	30-31
2.9. Subscribing to messages with the DMP	31
Handling Vertical Design	31

Purpose

This document provides general information on how to create content for the DMP 4310, supported file formats and optimization recommendations.

Flash content creation for the web, which runs on a desktop computer, is not the same as content creation for the device.

Content created for the device needs different approach and optimization techniques.

This guide provides optimization recommendations and basic scripting information. Provided links and information for AS2, JS and AS3 will help content developers in repurposing their content for the DMP 4310 if it was initially created by using different from AS2 coding.

Targeted Audience

Cisco Partners, creative agencies, interested parties etc.

Prerequisites

Basic programming skills, knowledge of Flash ActionScript.DMP Supported Formats

DMP 4310 Supported Formats

Supported on DMP 4310					Not Supported on DMP 4310
Swf compiled ActionScript 2 (and 1)					Webpages
supported in the native video decoding	MPEG-1				JavaScript
	MPEG-2 @HL				Audio in SWF files
	H.264 for Main Profile @ Level 4.0 and High Profile @ Level 4.0				
	WMV9 MP@HL				Any playlist asset that uses ActionScript 3
video stream bit rates MAX. recommended		MPEG-2	MPEG-4.10	WMV9	8-bit alpha channel video
	HDTV	16 Mbps	12 Mbps	10 Mbps	
	SDTV	16 Mbps	12 Mbps	10 Mbps	RTP elementary streams that send audio and video to different ports
in the native audio decoding	Mpeg-1 Layer 2		75 Mbps min. bandwidth		FF, Rewind, Pause (trick play) with RTP
	Dolby AC-3 Audio support		29 Mbps min. bandwidth		JPEG backgrounds
IGMPv3 for multicast					Any SWF that uses the "Stage.align" property
RTP using port 7777, by way of DMM, DMPDM, API, and ActionScript API					
Our RTP support is specifically MPEG-TS/RTP/UDP multicast, also called TS over RTP over UDP					
PNG, GIF, JPEG, PNG recommended.					

Must pass Flash variables to Flashlite by appending a special parameter to the URI for any SWF. Syntax is:

http://HOST/PATH/SWF?fl_flashvars=HTTP_ENCODED_VALUE

(In DMM-DSM, this requirement applies to GoToUrl and DMP Failover Retry)

Supported Bit rate up to 16 Mbit/sec	SD: 3 Mbit/sec - 5 Mbit/sec
	HD: 12 Mbit/sec - 16 Mbit/sec
h.264 in MPEG-2 container recommended for	SD: Approximately 5 Mbit (Min2 Mbit)
	HD: Approximately 6 Mbit (Min3 Mbit)
PRESETS EXAMPLE FOR 1080P 30	
Quality	Best; TV standard: NTSC
Frame size	29.97 drop frame
Field Order	None (progressive)
Pixel aspect ratio	Square Pixels
Profile	Main
Level	High level
Bitrate Encoding	CBR, or VBR (for VBR numbers should not exceed Max bitrates specified above)

Common source program resolutions and frame refresh rates:

- 480, 544 and 640 x 480i @ 25, 29.97 and 30Hz
- 480, 544 and 640 x 480p @ 23.976, 24, 29.97, 30, 59.94 and 60Hz
- 704 and 720 x 480i @ 29.97 and 30Hz
- 704 and 720 x 480p @ 23.976, 24, 29.97, 30, 59.94 and 60Hz
- 704 and 720 x 576i @ 25Hz
- 704 and 720 x 576p @ 23.976, 24, 25 and 50Hz
- 1280 x 720p @ 23.976, 24, 25, 29.97, 30, 50, 59.94 and 60Hz
- 960, 1280, 1440 and 1920 x 1080i @ 25, 29.97 and 30Hz
- 960, 1280, 1440 and 1920 x 1080p @ 23.976, 24, 25, 29.97 and 30Hz

VIDEO RESOLUTIONS (DMP 4310G)

	MINIMUM	MAXIMUM
MPEG-2	16x32	1920X1080

<i>MPEG-4.10 (H.264)</i>	64x64	1920X1080
<i>MPEG-1</i>	64x64	1920X1080
<i>WMV9</i>	64X64	1920X1080

- Generally, **outputs for 720p and 1080p require the “High” profile** setting due to their frame sizes. Using a profile of “Normal” or lower will generate a “Mismatch” error message.
- **Bitrates:** Testing has shown that a bitrate range between 8Mbps-16Mbps yield visually acceptable results for DMP playback. Above or below this range emphasize image alterations. Experimentation in bitrate range and codec application may be necessary. Since software component updates, and code changes may be injected into the workflow can occur at any time, testing the result is always recommended prior to deployment.
- **A square pixel** aspect ratio is recommended for DMP playback.
- In Adobe Premiere, it’s best to use existing presets for this project:
When you open a new project, please choose Adobe HDV 720p 30 or HDV 1080p 30.
And then, in Export Adobe Media Encoder please choose MPEG2 for format, Range: render entire sequence, Preset: Custom; Video Codec: MainConcept MPEG Video stream)
Multiplexing:TS (Transport Stream).

Content Examples

Videos

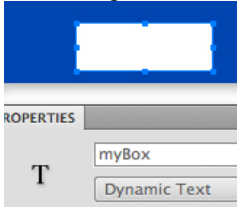
Setup basic application in Adobe Flash Professional.

Create new AS 2.0 file, set document size to 1366x768. Create a Dynamic Text instance; call it *myBox*, set size to 1013x570 (or your preferred size). In the “properties” select the “Show border around text” option.

Insert a new layer, call it Scripts. Open AS 2.0 script file (shortcut-hit F9), include as first line in your script file:

```
#include "videodemo.as"
```

We will explain file “videodemo.as” in 1.2. Creation of the video playlist chapter.



II. Setup includes directory and structure:

Copy folder com with to the project directory. Subfolders structure looks like this: com/Cisco/dms/MediaPlayer.as
MediaPlayer.as file includes necessary API.

1.1. Develop code for playing .mpg through flash:

Create new file “videodemo.as” in the same directory as Flash Project file. Insert following code in it:

```
import com.cisco.dms.MediaPlayer; // (1)

var mp:MediaPlayer = new MediaPlayer(); // (2)

mp.setColorKey(0x01, 0x00, 0x01, 0); // (3)

myBox.backgroundColor = 0x010001; // (4)

mp.playVideoToObject("http://...../your_movie.mpg", true, myBox); // (5)

mp.subscribe(); // (6).
```

First line (1) imports DMP API, so we could create instance of MediaPlayer class (2) and use its methods.

At line (3) we set the color key to instance of MediaPlayer, which determines the area for the MPG file to play.

At line (4) we set the **same color** to the myBox variable, created in section 1. It is important; otherwise your video won't be playing properly.

At line (5) we use function playVideoToObject(path, loop, object) to set MediaPlayer for playing. First parameter path – must be network path to the .mpg file. Second parameter, loop, is to specify whether we want video loop or not; and must be “true” or “false”. Third parameter, object, points to the object in .fla, to which we want to play .mpg (myBox in our case). Command at line (6) subscribes for receiving events from MediaPlayer.



Reference Template:

1.1._PlayingSimpleVideo

1.2. Creation of the video playlist

To play several videos in certain order, create myBox flash object, like in previous example. We will need to add code, which allows tplaying videos to the myBox object.

Below is an example of action script file “videodemo.as”, which needs to be included into presentation (don’t forget to include it in your .fla):

```

import com.cisco.dms.MediaPlayer; // (1)

var mp:MediaPlayer = new MediaPlayer(); // (2)
mp.stopVideo();
mp.setColorKey(0x01, 0x00, 0x01, 0);

var playlist:Array = new Array("http://.../movie1.mpg", "http://.../movie2.mpg"); // (3)

var playlistIndex:Number = 0;
var videoStarted:Boolean = false; // (4)
myBox.backgroundColor = 0x010001;

function playNext():Void { // (5)
    playlistIndex = (playlistIndex+1) % playlist.length;
    mp.playVideoToObject(playlist[playlistIndex], false, myBox);
};

var listener:Object = new Object(); // (6)
listener.videoStatus = function(msg:String)
{
    if(msg == "START") {
        videoStarted = true;
    } else if (msg == "STOP") {
        if(videoStarted)
            playNext();
        videoStarted = false;
    } else
        mp.logInfo("Failure...");
};

mp.addListener(listener);
mp.playVideoToObject(playlist[playlistIndex], false, myBox);
mp.subscribe();

```

This code plays an array of videos, one by one, in an infinite loop.

Like in previous example in first line (1) we import MediaPlayer class, instantiate and initialize it in (2). In line (3) we create an Array of movie links, in the order we want these movies to play.

Function playNext(): is to start each next movie in the list (5). We also create a listener (6), which will call that function upon receiving the message from MediaPlayer.



Reference Tmplate:

1.2. *VideoPlaylist*

1.3. Creating output box for MediaPlayer on the fly.

Instead of creating output box for MediaPlayer inside the Flash like we did with in previous example (myBox), we also can create it “on the fly”. We need to add the following code to videodemo.as:

```

var playlistX:Number = 0;
var playlistY:Number = 0;
var playlistWidth:Number = 1013; // (1)
var playlistHeight:Number = 570; // (2)

var myBox:MovieClip = _root.createEmptyMovieClip("tvBox", _root.getNextHighestDepth());
with(myBox) {
    moveTo(0,0);
    beginFill("#000000", 100);
    lineTo(playlistWidth, 0);
    lineTo(playlistWidth, playlistHeight);
    lineTo(0, playlistHeight);
    lineTo(0, 0);
    endFill();
};
myBox._x = playlistX;
myBox._y = playlistY;

```

The code above creates rectangle myBox with width 800 and height 600.



Reference Tmplate:

1.3. *CreationOutputBoxOnFly*

1.4. How to set correct path to .mpg files.

In previous examples we were setting absolute path's to .mpg files.

If you play files from your server, which may be Apache server or IIS, the array of movies for MediaPlayer might look like this:

```

var playlist:Array = new Array(
    "http://XX.XXX.XXX.XXX/movies/mpeg1.mpg",
    "http://XX.XXX.XXX.XXX/movies/mpeg2.mpg");

```

Where XX.XXX.XXX.XXX is IP address of you machine, and “movies/mpeg2.mpg” is relative path from document folder on your server.

In case your server is in certain port, for example 8888, then pathes in the movie array must be looking like that:


```
var playlist:Array = new Array(
    "http://XX.XXX.XXX.XXX:8888/movies/mpeg1.mpg",
    "http://XX.XXX.XXX.XXX:8888 /movies/mpeg2.mpg");
```

The other option is to play your application from DMP, what you need to do – is copy your files to DMP ftp server. In that case, you're going to play files "locally", and array of movies for MediaPlayer will be:

```
var playlist:Array = new Array(
    "file:///tmp/ftproot/usb_1/movies/mpeg1.mpg ",
    "file:///tmp/ftproot/usb_1/movies/mpeg2.mpg ");
```

It is possible to make you application IP-flexible, i.e. dynamically determine IP address of server, where .swf is located. Below is an example of how to do this:

```
var my_lc = new LocalConnection();
var prefix = "http://" + my_lc.domain();
var playlist:Array = new Array(
    prefix + "/movies/mpeg1.mpg",
    prefix + "/movies/mpeg2.mpg");
```

Code above demonstrates how we detect domain automatically and substitute with it hardcoded IP. In case you have your web server on a port, you need to add it to path variable. In that case prefix variable will look like this:

```
var prefix = "http://" + my_lc.domain()+ ":8888";
```



Reference Tmplate:

[1.4. _ PlayistFromLocalStorage](#)

1.5. Decreasing intervals between movies in the list.

Short intervals in between movies in the playlist are unavoidable, but there's a way to reduce them programmatically. In order to do this we need to set parameter stay to 1 for each movie in the list, except the last one. Setting that parameter to 1 will make the last frame of the movie to stay on the screen a bit longer, while next movie is loaded. The playlist array will look as follows:

```
var playlist:Array = new Array(
    "http://XX.XXX.XXX.XXX/movies/mpeg1.mpg?stay=1",
    "http://XX.XXX.XXX.XXX/movies/mpeg2.mpg?stay=1",
    "http://XX.XXX.XXX.XXX/movies/mpeg3.mpg?stay=1",
    "http://XX.XXX.XXX.XXX/movies/mpeg4.mpg?stay=1",
    "http://XX.XXX.XXX.XXX/movies/mpeg5.mpg?stay=1",
    "http://XX.XXX.XXX.XXX /movies/mpeg6.mpg?stay=0");
```

Remember, that the last movie in the list must end with stay parameter equal to 0.



Reference Tmplate:

[1.5. _DecreasingIntervals](#)

1.6. Playing movies from XML file list.

Sometimes it is convenient to have a list of .mpg links in separate file, and load them dynamically. For instance, Create file videos.xml and add following lines:

```
<item>http://XX.XXX.XXX.XXX/movies/mpeg1.mpg</item>
<item>http://XX.XXX.XXX.XXX/movies/mpeg2.mpg</item>
<item>http://XX.XXX.XXX.XXX/movies/mpeg3.mpg</item>
```

File videodemo.as is going to look as follows:

```
import com.cisco.dms.MediaPlayer;
var mp:MediaPlayer = new MediaPlayer();
mp.stopVideo();

var videoStarted:Boolean = false;

var playlist:Array = new Array(); // (1)
var playlistXML:XML = new XML(); // (2)

var playlistIndex:Number = 0;
mp.setColorKey(0x01, 0x00, 0x01, 0);
myBox.backgroundColor = 0x010001;

function playNext():Void {
    playlistIndex = (playlistIndex+1) % playlist.length;
    mp.playVideoToObject(playlist[playlistIndex], false, myBox);
};

var listener:Object = new Object();
listener.videoStatus = function(msg:String)
{
    if(msg == "START") {
        videoStarted = true;
    } else if (msg == "STOP") {
        if(videoStarted) {
            playNext();
        }
        videoStarted = false;
    } else {
        mp.logInfo("what just happened...");
    }
};
mp.addListener(listener);

playlistXML.ignoreWhite = true;
playlistXML.onLoad = function(success:Boolean) { // (3)
```

```

    if(success) {
        for(var i = 0; i<playlistXML.childNodes.length; i++) {
            playlist[playlist.length] =
unescape(playlistXML.childNodes[i].firstChild.nodeValue);
            mp.logInfo(playlist[i]);
        }
    } else {
        playlist[0] = "http:// http://...../your_movie.mpg";
    }
    mp.playVideoToObject(playlist[playlistIndex], false, movieBox); //(4)
    mp.subscribe();
};
playlistXML.load("videos.xml");// JavaScript Document // (5)

```

As you may have noticed, the first part of the script is similar to one at 1.2, except we do not fill array playlist with links to videos; we create an empty array (1) instead. At line (2) we create XML object, which later, at (5), we use for loading XML from file ("videos.xml"). Upon loading XML it gets parsed (3) and array playlist gets filled with the link strings. We play the video to the MediaPlayer by using the same playVideoToObject() API at line (4).



Reference Tmplate:

[1.6. XML Playlist](#)

XML

Loading Data from static XML

One of the helpful techniques is dynamic uploading of data from XML file. Flash has build in XML parser, so it is very easy to access XML data from Action Script. In following example we demonstrate how to read and dynamically rotate XML data.

There are four Dynamic Text objects created on the stage, with instances, named "status_1", "status_2", "status_3", and "status_4" respectively.

```

var m_index:Number = 0;
var interval:Number;
var myxml:XML = new XML(); // (1)
myxml.ignoreWhite = true;
myxml.onLoad = function(success:Boolean):Void{
    if(success == true){
        m_index = 0;
        poexali();
        setInterval(data_loading, 1000); // (2)
    }
};

function data_loading(){
    var messages:XMLNode = myxml.firstChild; // (3)
    if(m_index >= messages.childNodes.length-1)
        m_index = 0;
}

```

```

var msg:XMLNode = messages.childNodes[m_index];           //(4)

_root.status_1.htmlText = msg.childNodes[0].firstChild.nodeValue;
_root.status_2.htmlText = msg.childNodes[1].firstChild.nodeValue;
_root.status_3.htmlText = msg.childNodes[2].firstChild.nodeValue;
_root.status_4.htmlText = msg.childNodes[3].firstChild.nodeValue;

m_index++;                                             //(5)
}

myUrl = getProperty("", _url);                          //(6)
var m_index:Number = myUrl.lastIndexOf("/");
if(m_index > 0) myUrl = myUrl.substring(0, m_index+1);
myUrl += "data.xml";
myxml.load(myUrl);                                     //(7)

```

In line (1) variable of class XML is created, and function “data_loading” is called (2) at the onLoad event. Variable of class XMLNode is created at (3), and main first child of XML structure is assigned to it. It is easy to get an array of child nodes in AS 2, we do it using childNodesoperator. At line (4), we access elements of that array, using index (5), which we increment each function call. At line (6) and further we show how to determine correctly location of data file (“data.xml”), and at line (7) we are loading data to xml. Data xml file should be located in the same place as .fla file, and below is example of its content:

```

<?xml version="1.0" encoding="utf-8"?>
<messages >
  <message >
    <item>Sceduled</item>
    <item>Arrived</item>
    <item>Departed</item>
    <item>At The Gate</item>
  </message>
  <message >
    <item>At The Gate </item>
    <item>Departed </item>
    <item>At The Gate </item>
    <item>Departed </item>
  </message>
</messages >

```

That XML file could contain as many messages as needed.



Reference Tmplate:

[*LoadingDataFromStaticXML*](#)

Tickers *DMD Capable of**Typewriter Ticker**

It is a good idea to include a ticker into your sign to emphasize important information or to bring up a message. There are different approaches, and typewriter ticker is the one, we recommend for DMP box. It does not require much of redrawing, and is much easier for user to read, since main text is static and new letters appear one by one. Below is a code snippet for typewriter example. You need to create on stage Dynamic Textbox and call it's variable "output".

```

var maxNum = 200;
var interval = 20; // (1)
var initText = "";
var initTextLen = initText.length;
var counter = 0;
var outputText = "";

function printNextChar() { // (2)
    var wordChar = "";
    var wordCharLen = 0;
    var end_of_line = false;
    if (counter < initTextLen) {
        while ((counter + wordCharLen < initTextLen) && (wordChar != " ")) {
            wordCharLen++;
            wordChar = initText.charAt(counter + wordCharLen);
        }
        if ((initText.charAt(counter + wordCharLen) + initText.charAt(counter + wordCharLen+1))
            == "\n") {
            end_of_line = true;
            break;
        }
    }
    if (!end_of_line) {
        if ((outputText.length + wordCharLen) < maxNum) {
            outputText += initText.charAt(counter);
            set ("output", outputText);
        }
        counter++;
    }
}

} else { // (3)
    set ("output", "");
    counter = 0;
    wordChar = "";
    wordCharLen = 0;
    outputText = "";
    end_of_line = false;
}

}

setInterval(printNextChar, interval); // (4)

```

Function `printNextChar()` at line (2) is called from (3) with interval, specified in (1). It prints characters from string one by one, and clears (4), when the whole message is printed.

Flexibility is very important for signage, and one of the creative approaches is dynamic uploading of data from XML file. That way the demo content can be changed without building new .swf, all users need to do – just replacement of the Message in XML file.

Here is the code, you may want to add to your ticker code in order to read the ticker text from XML:

```
ticker_xml = new XML();
ticker_xml.load("ticker_text.xml");
ticker_xml.onLoad = function(){
    myroot = new XML;
    myroot = this.lastChild.previousSibling;
    if(myroot.nodeName.toLowerCase() == "text") {
        initText = myroot.firstChild.nodeValue;
        initTextLen = initText.length;
    }
}
```

Create a file "ticker_text.xml" and put it into the same folder as your .FLA file. Content of XML file will look like following:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<text> This is an example of typewriter ticker, make it as long as you wish and as quick as you wish.</text>
```



Reference Template:

Typewriter_Ticker

Fade-out ticker

Definitely, the most eye-catching tickers are the ones with fade-out effect. To implement this, you need to do some tricky stage job. Create a dynamic text, call the instance `newsText`; convert it into Symbol: MovieClip. Call the instance `newsMC`. In order for fade-out effects to work, you need to embed fonts (click on "Embed.." button on DynamicText "Properties" panel, chose fonts you need and click OK).

Put following code on the Action page:

```
var counter:Number = 0;
var delay:Number = 100;
var xmlData:XML = new XML();
var mainElement:XML = new XML();
newsMC.newsText._alpha = 0;

function fadeout() { // (1)
    if (newsMC.newsText._alpha <= 0) {
        newsMC.newsText._alpha = 100;
        if (counter >= mainElement.childNodes.length)
            counter = 0;
        newsMC.newsText.text = mainElement.childNodes[counter++].firstChild.nodeValue; // (2)
    } else
        newsMC.newsText._alpha -= 5; // (3)
}
```

```

xmlData.ignoreWhite = true;

xmlData.onLoad = function(){
    mainElement = this.firstChild;
    setInterval(fadeout, delay);           //(4)
}
xmlData.load("ticker.xml");             //(5)

```

In fadeout()function (1), code loads the ticker strings from "ticker.xml" file at (5) and sets interval for fadeout() function (4) which decreases alpha parameter of the ticker text (3) and loads new string from XML (2), when alpha of the text turns zero.



Reference Template:

Fade_Out_Ticker

Clocks

Digital Clock

As you may know, DMP 4310 has it's own NTP service, which allows to include accurate clocks in the signs. In order to use it, we have to turn the NTP service ON in the DMP-DM.

You can set the time zone and refresh interval, and change NTP host name, if necessary.

```
var today:Date = new Date(); // (1)
```

```
var dateString:String = today.getFullYear().toString()+ "/" +
(today.getMonth()+1).toString()+ "/" + today.getDate().toString(); // (2)
```

```
var timeString:String = today.getHours().toString()+ ":" + today.getMinutes().toString()+ "." +
today.getSeconds().toString(); // (3)
```

```
time = timeString; // (4)
```

```
date = dateString; // (5)
```

At line (1) we create variable of class Date, and then use it, constructing date string (2) and time string (3). These strings are assigned to Dynamic Text variables at (4) and (5).



Reference Tmplate:

Digital_Clock

Image Slideshow *DMD Capable of

Adding alpha transition

We recommend using transition effects between pictures in slideshow sparingly, meeting general recommendations for small devices. In following example we demonstrate how to load pictures from external text file and rotate them with fading out effect.

```
var image_index:Number = 1;
var mIndex:Number;

var lv:LoadVars = new LoadVars(); // (1)
lv.load("data.txt");

lv.onLoad = function(success:Boolean):Void{ // (2)
    if(success == true){
        mIndex = parseInt(this.maxIndex);
        initImage();
    }
};

function initImage(){ // (3)
    _root.imageHolder._xscale = 100;
    _root.imageHolder._yscale = 100;
    _root.imageHolder._alpha = 100;
    _root.imageHolder._x = _root.mask_mc._x;
    _root.imageHolder._y = _root.mask_mc._y;
    _root.imageHolder.loadMovie(lv["image1"]); // (4)
    setTimeout(decreaseAlpha, 1500); // (5)
}

function loadImage():Void{

    if(_root.imageHolder._alpha <= 0){
        image_index++;
        if(image_index > mIndex){
            image_index = 1;}

    _root.imageHolder.loadMovie(lv["image" + image_index]); // (6)
    _root.imageHolder._alpha = 100;
    setTimeout(decreaseAlpha, 1500); // (7)
}
```



```

    } else {
        decreaseAlpha();
    }
}

function decreaseAlpha():Void{
    _root.imageHolder._alpha -= 1; // (8)
    setTimeout(loadImage, 10); // (9)
}

```

At line (1) we show how to load images from into instance lv of class LoadVars. Image loading starts upon text file is read (2), and function InitImage() is called. Each time, new image is loaded (4) and (6), image will stay for one and a half seconds (7) and (9), then gradually fadeout. In order to make that effect we decrease Alpha by 5 (at 8) every 50 milliseconds (9).

Place "data.txt" with following content in the same directory as your .FLA file:

```

&maxIndex=7
&image1=images/image1.jpg&
&image2=images/image2.jpg&
&image3=images/image3.jpg&
&image4=images/image4.jpg&
&image5=images/image5.jpg&
&image6=images/image6.jpg&
&image7=images/image7.jpg&

```

Images must go in the "images" directory, and file names must correspond ones in "data.txt" file.



Reference Tmplate:

Image_Slideshow

RSS Feeds *DMD Capable of

What is RSS?

RSS (abbreviation for Really Simple Syndication) an XML-based format, developed for publishing frequently updated works, such as news, weather, blogs. Here is an example of RSS format:

```

<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
    <title>RSS Title</title>
    <description>This is an example of an RSS feed</description>
    <link>http://www.someexamplerssdomain.com/main.html</link>
    <lastBuildDate>Mon, 06 Sep 2010 00:01:00 +0000 </lastBuildDate>
    <pubDate>Mon, 06 Sep 2009 16:45:00 +0000 </pubDate>

    <item>
        <title>Example entry</title>
        <description>Here is some text containing a description.</description>
    </item>
</channel>
</rss>

```

```

<link>http://www.wikipedia.org/</link>
<guid>unique string per item</guid>
<pubDate>Mon, 06 Sep 2009 16:45:00 +0000 </pubDate>
</item>

</channel>
</rss>

```

Each RSS stream should contain root element <rss>, channel element <channel>, which usually provides general publisher information, such as Company name, etc., and one or more item elements <item>, which contain broadcasting units, such as news, or blog entries.

Here is an example of live RSS from Cisco :

http://www.cisco.com/assets/cdc_content_elements/rss/whats_new/whatsnew_rss_feed.xml

You could look at View->Source in your browser to see raw formatted XML of that RSS feed.



Reference Tmplate:

RSS_feed

How to embed RSS feed into my sign?

Action script has very good build-in tools for working with XML, and we show you how to do this.

Following code loads RSS from `RSS_path`, which is path to proxy (going to be discussed later)

```

var updateRSS = function() {
XML_var = new XML(); // (1)
XML_var.load(RSS_path); // (2)
XML_var.onLoad = loadRSSNews; // (3)
}

```

At line (1) we create new XML object, to hold loaded RSS XML, loading XML (2), and calling parsing function upon loading (3). Parsing function `loadRSSNews` is needed to extract information from RSS stream and place it to certain place on the screen.

```

function loadRSSNews()
{
mainTag = new XML;
itemsList = new Array;
elementsList = new Array;
mainTag = this.firstChild.nextSibling; // (1)

itemsList = mainTag.childNodes[0].childNodes; // (2)
RssText = " ";
RssTitle = " ";

Items = new Array;

for (var i = 0; i < itemsList.length; i++)
{
if (itemsList[i].nodeName.toLowerCase() == "item") // (3)

```

```

    {
        var nItem = new Item(); // (4)
        elementsList = itemsList[i].childNodes; // (5)

        for (j = 0; j < elementsList.length; j++)
        {
            if (elementsList[j].nodeName.toLowerCase() == "title") // (6)
                nItem.setTitle (elementsList[j].childNodes[0]);
            if (elementsList[j].nodeName.toLowerCase() == "description") // (7)
                if (elementsList[j].childNodes[0] != null)
                    nItem.setText (elementsList[j].childNodes[0]);
        }
        Items.push(nItem); // (8)
    }

    }
    RssText = Items[index].getText(); // (9)
    RssTitle = Items[index].getTitle();
    index++;
    if (index >= (Items.length-1))
        index=0;
}

```

At line (1) of the code above we getting main tag of the XML string, using “this” notation, which refers to RSS XML object we have created. We getting <rss> node, and then getting to <channel> node, using .nextSibling AS function. You may want to look at RSS structure, provided in 3.1.1, as an illustration. In line (2) we fill array itemList with elements from <channel>. In most cases copyright policies of RSS provider requires to expose all information, contained in RSS stream with no exceptions. To shorten our example we will show just how to show only item content, its title and description. In line (3) we loop through all itemList elements, and going to work only with those, whose tag name is “item”. In lines (4) and (5) inside the cycle we create a new object to hold item info (nItem) and new array (elementsList), at this time – to hold elements of each item. Then we again loop through all item elements and grab only those with name “title”(6) and “description”(7). We set properties of nItem with what we got from “title” and “description” and add it to Item array at (8). At line (9) we set variables of Dynamic Text RssText and RssTitle to values of Items elements on rotation basis.

Overcoming cross-domain issues with proxy.

Although we could easily see RSS XML in the browser, we could not as easy get it into Flash application because of the security limitations. In order to load XML data from different domain you need additional “helper” script, which we call “proxy”. Following proxy code is written on PHP:

```

<?php
$url = "http://YOUR_RSS_LINK";
$session = curl_init($url);

curl_setopt($session, CURLOPT_HEADER, false);
curl_setopt($session, CURLOPT_RETURNTRANSFER, true);

$xml = curl_exec($session);
header("Content-Type: text/xml");

```

```

echo $xml;
curl_close($session);
?>

```

Save this code in “proxy.php” file and put in the same folder as your flash application. You need to include in your action script path to this proxy.

```
var RSS_path = "proxy.php";
```

Variable `RSS_path` is used to load RSS in our example in 3.1.2 line (2).

NOTE: in order for php to work, please configure `php.ini` file on your server. It is usually located in PHP folder. Line in `php.ini`

```
extension=php_curl.dll
```

must be not commented.

Database

Flash – Database communications

There’s no direct way to connect Flash to Database, and an intermediary application is needed between them. There are three ways to implement this architecture:

I. The XMLConnector component:

Use the XMLConnector component to connect to an external XML document to bind to properties in the document. Its purpose is to read or write XML documents by using HTTP GET operations, POST operations, or both. It acts as a connector between other components and external XML documents.

II. **Combination of the XML class on the Flash side with a CGI application on the server side.** This method uses a POST HTTP request to connect to the application server. For a good explanation of this, see Flash help (ActionScript reference guide > Working with external data > Sending and loading variables to and from a remote source > Using the XML class).

III. **Using XMLSocket class.** The actual database operations are performed by a backend Java application that communicates with the database through JDBC protocol. Flash would communicate with this intermediary application through sockets.

Scripting

What is ActionScript

The main features of ActionScript 2.0 include the following:

Object-oriented programming (OOP) model. The primary feature of ActionScript 2.0 is a model for creating object-oriented programs. ActionScript 2.0 implements several object-oriented concepts and keywords such as class, interface, and packages.

The OOP model provided by ActionScript 2.0 is a "syntactic formalization" of the prototype chaining method, used in previous versions of Flash to create objects and establish inheritance. With ActionScript 2.0, you can create custom classes and extend Flash's built-in classes.

Strict data typing ActionScript 2.0 also lets you explicitly specify data types for variables, function parameters, and function return types. For example, the following code declares a variable named `userName` of type `String` (a built-in ActionScript data type, or class): `var userName:String = "";`

Compiler warnings and errors Compiler provides warnings and error messages that help you find bugs in your applications faster than was previously possible in Flash.

Value of AS2

ActionScript 2.0 easily integrates database systems, interactive programming languages (php, asp, etc.), and user interactivity. It is stable when computing values, algorithms, and algebraic expressions.

When designing for small appliances such as mobile devices, DMP, etc. it is important to use tools, which are lightweight, and allow maximum optimization.

AS2 gives flash developers freedom of code with the benefit of speedy production.

AS2 allows provide lighter programming environment.

The line of our products supports Flash AS2. It is cross compatible with 4400G and templates developed for 4400G.

Reference Information:

http://www.adobe.com/devnet/flash/articles/as_bestpractices.html

AS 2 vs.JS

AS2 and JS are very similar; they are both compatible with ECMA-3. ECMAScript is widely used for client-side scripting on the web, in the form of several well-known dialects such as JavaScript, JScript, and ActionScript.

If you have a framework of code in JS and you want to use their logic in AS2 it is possible. Therefore, JavaScript content, developed for 4400G can be repurposed for 4310.

The following code, which works in any compliant player(4400G and 4305G), creates a text field at depth 0, at position (0, 0) on the screen (measured in pixels), that is 100 pixels wide and 50 high. Then the text parameter is set to the "Hello!" string, and it is automatically displayed in the player:

```
createTextField("message", 0, 0, 0, 100, 50);  
message.text = "Hello!";
```

Described above code in ActionScript 2.0 looks like this:

```
{  
    var txtHello:TextField = this.createTextField ("txtHello", 0, 0, 0, 100, 50);  
    txtHello.text = "Hello!";  
}
```

On DMP Flash Player produced better results in comparison with the browser.

Please refer to examples Small Touch Screen with 3 Buttons Flash and same presentation written in JS Small Touch Screen with 3 Buttons JSin our content library:www.in.cisco.com/etg/digitalmedia/selling_resources/demo_downloads.shtml.

AS 2 vs. AS 3

The AS3 syntax is very similar to Java's, and there are all the object-oriented features you're familiar from Java, such as extending classes and implementing interfaces.

Is AS3 always the best choice?

Developers use different approach for various applications. For example, Flash AS3 is great for the web applications, which designed to run on computers. In fact, AS3 brings a lot of overhead on small devices, including DMP.

It really depends on what you want to accomplish, as programming languages are just tools.

Logical organization of basic packages <http://www.adobe.com/livedocs/flex/2/langref/package-summary.html> allows quick and easy access to required functionality.

Some examples of code - level differences are explained below.

Compiler Directives:

#endinitclip, #initclip, #include do not exist in AS3

Global Properties:

Properties `_root`, `_accProps`, `_focusrect`, `_highquality`, `maxscroll`, `_parent`, `_quality`, `scroll`, `_soundbuftime`, `this` - in AS3 are replaced by other classes and properties (see specifications).

Properties `_level`, `_global` have been deprecated. The concept of levels does not exist in ActionScript 3.0. Instead, it provides direct access to the display list. See the `flash.display` package for details.

The closest equivalent to "`_root`" is the Stage, which serves as the root of the ActionScript 3.0 display list.

Operators:

Operators `add`, `eq`, `gt`, `ge`, `<`, `it`, `le`, `and`, `not or`, `ne` were removed in AS3, and "instanceof" operator is replaced by "is". In action script to following operators must be used:

concatenation (+) operator
equality (==) operator
greater than (>) operator
greater than or equal to (>=) operator
inequality (!=) operator
less than (<) operator
less than or equal to (<=) operator
logical AND (&&) operator
logical NOT (!) operator
logical OR (||) operator
inequality (!=) operator

Syntax:

AS2 uses a lot of attributes that start with an underscore. In AS3 the underscores are removed.

1. In AS2:

._x, *._alpha*, *etc.*

2. In AS3:

.x, *.alpha*

Function Return types are lowercased in AS3:

1. In AS2:

function():Void{}

2. In AS3:

function():void{}

Properties of Classes MovieClip and TextField:

In AS2 properties of MovieClips and TextFields that started with `_` (underscore).

In AS3 we have instead properties of DisplayObject class.

- In AS2:

_x, *_y*, *_alpha*, *_visible*, *_width*, *_height*, *_xscale*, *_yscale*

In AS3:

```
flash.display.DisplayObject.x,
flash.display.DisplayObject.y,
flash.display.DisplayObject.alpha,
flash.display.DisplayObject.visible,
flash.display.DisplayObject.width,
flash.display.DisplayObject.height,
flash.display.DisplayObject.scaleX
flash.display.DisplayObject.scaleY
```

Movie clip methods instead of functions:

In AS2 we used to create Movie Clip by using `holder.createEmptyMovieClip("name", depth)`; function. We need to instantiate a new class instead in AS3: `new MovieClip()`.

We also use `MovieClip()` class methods an AS3 instead of AS2 functions:

In AS2:

```
holder.duplicateMovieClip("name", depth)
```

In AS3:

```
holder.attachMovie("linkageName", "name", depth)
```

Global Functions for Timer, Intervals & Delays:

in AS3 moved to `flash.utils` package.

In AS2:

```
setTimeout()
setInterval()
clearInterval()
clearTimeout()
getTimer()
```

In AS3:

```
flash.utils.setTimeout()
flash.utils.setInterval()
flash.utils.clearInterval()
flash.utils.clearTimeout()
flash.utils.getTimer()
```

Global Function replaced with method of the Sprite class:

In AS3 there's no global function for dragging, we have method of Sprite class instead.

In AS2:

```
startDrag()
stopDrag()
```

In AS3:

```
flash.display.Sprite.startDrag()
flash.display.Sprite.stopDrag()
```

Global Function replaced by method of String class:

substring() in AS2 was replaced by method of String class String.substring().

In AS2:

substring()

In AS3:

String.substring()

Different Event Model. Adding and removing a listener:

In Action Script 3, the event model has changed. In AS3 event model, we do not use class-specific addListener() method, like in AS2.

In AS3 the class inherits the addEventListener() method from the EventDispatcher class.

1. In AS2:

addListener()
removeListener()

2. In AS3:

flash.events.EventDispatcher.addEventListener()
flash.events.EventDispatcher.removeEventListener()

Removed methods in BitmapData class.:

Methods attachBitmap() and attachMovie() do not exist in AS3.

Use addChild() to add child display objects instead.

Removed methods in AsBroadcaster class.:

initialize() Method removed in AS3. There is no direct equivalent in ActionScript 3.0, but similar functionality is achieved by subclassing the EventDispatcher class. For example, the DisplayObject class extends EventDispatcher, so all instances of the DisplayObject and DisplayObject subclasses are capable of sending and receiving event objects.

There is new class Sprite in AS3:

In PrintJob class, for example, addPage() Method is replaced by flash.printing.PrintJob.addPage(). In ActionScript 3.0, changed data types of parameters: First parameter target is a Sprite data type; second parameter printArea is a Rectangle data type; third parameter options is the new PrintJobOptions data type; and fourth parameter frameNum is an int data type.

In AS2:

addPage()

In AS3:

flash.printing.PrintJob.addPage()

Reference Information: Full list of AS2 and AS3 differences is available

<http://www.adobe.com/livedocs/flex/2/langref/migration.html>

What is Flash Lite?

Adobe Flash Lite 3.1, released in February 2009. Lite 3.1

Player publishes itself as a player capable of handling Flash 9 ActionScript 2.0 content only.

Adobe Flash Lite is a lightweight version of Adobe Flash Player, a software application published by Adobe Systems.

Adobe Flash Lite software is the Flash technology specifically developed for mobile phones and consumer

electronic devices.

Flash Lite dramatically accelerates the delivery of rich content and browsing and customized user interfaces.

With Flash Lite designers and developers now have a new level of expressiveness, efficiency, and interactivity for content creation for mobile devices.

DMP AS vs. JS API

Information below is regarding our DMP APIs, provided by Engineering:

Comparison of Javascript (JS) and ActionScript (AS) APIs

Some JavaScript API's do not have an equivalent ActionScript APIs as there are no equivalent features to match (browser related calls). The reverse is also true - some ActionScript API's do not exist in the JavaScript API Flash related calls). For example, JS call `tvMediaPlayer_browserClose(id)` does not exist in the AS API as there is currently no Browser. AS call like `setColorKey(red:Number, green:Number, blue:Number, range:Number)` does not exist in JS API because this feature is not supported by the Browser.

However most video functions have a 1:1 mapping. For example, AS call `stopVideo()` is equivalent to the JavaScript function `tvMediaPlayer_stop()` where both API's stop video. In JavaScript the function `tvMediaPlayer_play(url)` is equivalent to the ActionScript function `playVideoToCoordinates(url:String, loop:Boolean, fullscreen:Boolean, x:Number, y:Number, w:Number, h:Number)` where both API's play video to a location defined on the screen.

Link to Action Script Media Player Full API :

Avoid Common Mistakes

- Always Make Sure You Use ActionScript 2.0, When Developing Content for DMP 4310.
- When you use ActionScript 2.0, make sure that the publish settings for the FLA file specify ActionScript 2.0 (the default for Flash CS3 is ActionScript 3.0). Additionally, if you open an older FLA file that uses ActionScript 1.0 or ActionScript 3.0 and begin rewriting it in ActionScript 2.0, change the publish settings of the FLA file to ActionScript 2.0. If you don't, your FLA file will not compile correctly, and errors won't be generated.
- When you create a new Flash CS4 document, you must specify Flash File (ActionScript 2.0), if you developing for DMP 4310.
- When you open an existing document, Flash examines the Publish Settings to determine which set of components to use. You have to make sure, that scripting language is ActionScript 2.0, if you want it to use for 4310.
- For 4310 playback ensure that in publish settings of your movie you choose Flash Player 9 and ActionScript 2.0 even if there is no script in your movie.
- Do not mix ActionScript 2.0 and ActionScript 3.0 components. You must use one set or the other for a given application.
- Avoid using components. Components are heavy. Use movie clips instead.
- Use var notation when defining variables
- Use block comments (`/* and */`) for multiline comments and single-line comments (`//`) for short comments. You can also use a trailing comment on the same line as the ActionScript code if necessary.
- Avoid attaching AS to objects. ActionScript code that is attached to objects is considered to be a poor coding style. ActionScript code that is attached to objects is difficult to locate, edit and debug. ActionScript code that is written on a timeline or in classes is more elegant and easier to build upon.

Content Tuning

Tweening and Transparencies

- Avoid tweening too many items simultaneously. Reduce the number of tweens and/or sequence

animation so that one begins when another ends.

- Transparency and gradients are processor intensive “expensive” tasks and should be used sparingly. Use png, or jpeg images Jpegs created in Photoshop instead of creating transparent objects in flash.
- Avoid expensive operations such as hiding objects using *visibility=""*, animated alpha transparency, masking, using the drawing API to create on-the-fly vector shapes, vector gradients, etc. Experiment with replacing expensive operations with other possible “inexpensive” tricks.

General Rules for Content Optimization

- Test your application on a DMP device as early and as often as possible.
- There is yet no way to use a desktop computer to realistically emulate the limitations in processor speed, heap memory, and other functions of a DMP.
- Test adding/removing animated elements to weigh their impact on performance.
- Evaluate content to determine whether animation/interaction can be easily achieved with the timeline, or simplified and made modular using ActionScript. It may be more efficient to use different movie clips rather than the timeline.
- Delete unused objects to optimize memory use.
- Low motion Flash objects (files with a .swf file extension) must be created with ActionScript 2.0 built for FlashLite 3.1
- Max recommended frame rate is 24 fps.
- The DMP 4310G supports both 1920 x 1080 and 1366 x 768 resolutions.
- Max recommended image size is 200Kb.
- For interactivity enhancements use animations sparingly.
- Keep your movie as small as possible. Get rid of content and code you don't need. Eliminate unnecessary code loops; the fewer frames, the better. Get rid of empty frames and layers.
- Reuse objects whenever possible.
- Use MovieClip symbols for elements and animations that are used more than once. Converting the animation/element into a movie clip has influence on the way the element is exported. It makes the exported SWF smaller than it would be when just reusing the element.
- Space your content naturally throughout your movie so that movie clips are initialized as they are used.
- Use scripted animation sparingly. Scripts with a huge amount of lines of code will be just as processor intensive as timelines with huge amount of frames of tweens. In other words, code and animations need to be concise.
- Simplify the code and animation. Keep it as simple as and as light as possible
- Use the “bitmap caching” feature for complex vector content, like complex gradients or text only when the bitmap can be generated once and then used without the need to update it.
- Avoid complex vectors with too many curves and points. Use graphics instead.
- Consider whether desired amount of effects is a good idea for device capabilities.
- Target your optimizations. It is important to focus on performance optimizations that are problems for your specific application
- Using bitmaps helps optimize rendering since flash player requires fewer processing resources on rendering pixels than vectors.

- Limit the types of line styles. Each line style has to be exported to the SWF file and this increases the size of the final movie.
- Prefer lines over brushes. Brushes increase the size of the exported SWF file. However, brush effects are usually displayed faster than line effects.
- Variable cannot use Flash reserved words, spaces, or symbols. Only letters, numbers, or underscores.
- Variable must begin with a letter.
- Always initialize variables before use. If not initialized, variable can contain garbage.

Action Script Optimization

- Eliminate unnecessary variables loops and function calls.
- Avoid creating variables and objects inside the cycle, try to create one before cycle and reuse it inside, if possible.
- Local variables preferable over global. Within functions, local variables are registered in a way that makes them much faster than global variables for the player to get and set.
- Don't overuse the Object type. Data-type annotations should be precise, because it improves performance. Use the Object type only when there is no reasonable alternative.
- Remove unnecessary “*trace()*” commands, or use Omit Trace Actions option when publishing your document.
- Use the ternary operator to write a short *if/else* statement. A ternary is used when assigning a value to a variable in a way that requires some conditional logic. Instead of

```

if ( c == d ) {
    x = a;
}
else {
    x = b;
}
you may want to use a simplified version:
x = ( c==d ) ? a : b;

```

- Use inline Array initialization


```
var myArray: Array = [“a”, “b”, “c”];
```

 Instead of creating instance using `new Array()`; and manually assigning content to it.
- Multiple variable/property assignments are also recommended:


```
myObject.x = myObject.y = 0;
a = b = c = 0;
```
- Avoid using loops whenever it is possible, straightforward coding is more preferable even if it does not look elegant. If it is impossible to avoid looping, keep the coding inside the loop as simple as possible. Focus on optimizing loops, and any repeating actions. Flash Player spends a lot of time processing loops (such as those that use the `setInterval()` function).
- Simplify hierarchy and structure of the package to objective minimum. Excessive inheritance (super class-subclass) adds unnecessary calls and additional processing.
- Use AS Garbage collection effectively. Make sure you delete each object which is no longer in use

explicitly and set local variables to null at the end of the block:

```
var myVar :Object = new Object();
/*some code here*/
delete myVar;
or
var x:String;
x=null;
```

- Use system function calls sparingly. For example each time you create Date object:


```
myDate = new Date();
```

 system time is called. To avoid unnecessary overhead, always make sure that timeouts between Date calls are not smaller than 500 milliseconds.
- Minimum recommended timeout for *setTimeout()* and *setInterval()* is 3000 milliseconds. Make time delays to work in asynchronous pattern, which will allow prevent functions from being called simultaneously.
- Avoid expensive operations such as hiding objects using *visibility=""*, animated alpha transparency, masking, using the drawing api to create on-the-fly vector shapes, vector gradients, etc. Experiment with replacing expensive operations with other possible “inexpensive” tricks.
- Avoid using *_alpha = 0* and *_visible = false* to hide onscreen movie clips. If you simply turn a movie clip’s visibility off or change it’s alpha to zero, it is still included in the player’s calculations, which can affect performance. Instead, remove movie clips completely off-stage or remove them using *removeMovieClip*. Alternatively, you can use blank keyframe.
- Remove event listeners from objects by using *removeListener()* before deleting or nullifying them. SWF data will not be removed from memory if any ActionScript functions are still referring to the SWF data when the movie clip is unloaded.
- Clear active intervals by using *clearInterval()* with *onUnloadMovie()* or *removeMovieClip()*.
- Optimize import statements. Avoid loading full libraries (*import library.*;*) load exact components instead (*import library.Component;*)

DMP-4310G API

The MediaPlayer API has been developed for the DMP 4310G. The interface should be included in Flash content that will need access to the platform API provided. To include this interface copy the "com" folder into the Flash project directory and add the following line to the Flash ActionScript code:

```
// Import MediaPlayer API
import com.cisco.dms.MediaPlayer;

// Create a MediaPlayer object
var mp:MediaPlayer = new MediaPlayer();
```

Calls may now be made on the "mp" object based on the documented interfaces. In previous chapter we already showed how to use *playVideoToObject()* function, and how to subscribe for event messages from MediaPlayer by using *subscribe()* method. Below we give you several examples of how to use other APIs.

2.1. Moving Video Box across the screen.

In public function *resizeVideoToObject(obj:Object);obj* – object to which video has to be resized to.

We can move the Video Box across the screen by using `resizeVideoToObject()` API function. Following function shows the example of how to change coordinates of the box in 5 second interval.

```
var updateBox = function() {
    if (myBox._x == 532){
        myBox._x = 132;
        mp.resizeVideoToObject(myBox);
    } else {
        myBox._x = 532;
        mp.resizeVideoToObject(myBox);
    }
}

//Move the box every 5 s
setInterval(updateBox, 5000);
```

You need to add this code to your “videodemo.as” script file.

2.2. Changing Volume

```
public function setVolume(volume:Number);
    volume - argument volume 0-100
public function getVolume();
    returns current volume of MediaPlayer
```

With the help of MediaPlayer API we are able to manipulate with the volume of playing movie. In following example volume of the movie played gradually goes up and down.

```
var vol = 0;
mp.setVolume(vol);

var flip = 1;
var updateVolume = function() {
    if (vol >= 100) flip = 0;
    if (vol <= 0) flip = 1;

    if (flip) {
        vol += 5;
    } else {
        vol -= 5;
    }
    mp.setVolume(vol);
}

setInterval(updateVolume, 500);
```

You need to add this code to your “videodemo.as” script file.

2.3. Manipulating with Video Brightness, Contrast and Saturation.

With the use of Media Player APIs we could easily change video settings programmatically.

2.3.1. Changing Brightness

```
public function setVideoBrightness (brightness:Number);
```

brightness - argument brightness 0-255

Function setVideoBrightness (brightness:Number) of MediaPlayer class sets the screen brightness to given value, which could be any number between 0 and 255. We must set it before calling playVideoToObject() function (1).

```
import com.cisco.dms.MediaPlayer;
```

```
var mp:MediaPlayer = new MediaPlayer();  
mp.setColorKey(0x01, 0x00, 0x01, 0);
```

```
myBox.backgroundColor = 0x010001;  
mp.setVideoBrightness(200); //(1)  
mp.playVideoToObject("http://...../your_movie.mpg", true, myBox);
```

```
mp.subscribe();
```

2.3.2. Changing Contrast

```
public function setVideoContrast(contrast:Number);
```

//contrast – argument contrast 0-255

Function setVideoContrast() is used to set video contrast prior to playing video. Argument is any number between 0 and 255

```
mp.setVideoContrast()(200); //(1)
```

2.3.3. Changing Saturation

```
public function setVideoSaturation(saturation:Number);
```

//saturation – argument saturation 0-255

Similar to previous examples setVideoSaturation() function is to set saturation to MediaPlayer before starting movie:

```
mp.setVideoSaturation(200); //(1)
```

2.4. Playing video URL to coordinates.

```
public function playVideoToCoordinates (url:String, loop:Boolean,  
fullscreen:Boolean, x:Number, y:Number, width:Number, height:Number);
```

//url - string, containing link to the movie

//loop - indication of whether we want to play video in loop or not

//fullscreen- indication of whether we want to play video full screen

//x - X coordinate of the video

//y - Y coordinate of the video

//width - width of the video

//height - height of the video

One of the useful MediaPlayer APIs is `playVideoToCoordinates()` function, which allows to play video to given coordinates. Following example shows how to use it. At first we play video to myBox object as usual (1), and after 5 seconds we move it (2) to coordinates (0,0).

```
import com.cisco.dms.MediaPlayer;
var mp:MediaPlayer = new MediaPlayer();
mp.setColorKey(0x01, 0x00, 0x01, 0);

myBox.backgroundColor = 0x010001;
mp.playVideoToObject("http://XX.XXX.XXX.XXX/movies/mpeg1.mpg", true, myBox);
//(1)

function moveBox():Void {
    myBox._x = 0;
    myBox._y = 0;
    mp.playVideoToCoordinates("http://XX.XXX.XXX.XXX/movies/mpeg1.mpg ",true,
        false, 0,0, 640, 480);
    // (2)
}

setTimeout(moveBox, 5000);
```

2.5. Playing video playback to coordinates.

```
public function resizeVideoToCoordinates (fullscreen:Boolean, x:Number, y:Number,
w:Number, h:Number);
```

```
//fullscreen - indication of whether we want to play video fullscreen
//x - X coordinate of video
//y - Y coordinate of video
//width - width of video
//height - height of video
```

Function `resizeVideoToCoordinates()` allows relocate and resize already playing video (playback). Usage is similar to previous example.

2.6. Setting and getting mib parameters of the DMP

Some of the internal parameters of DMP could be seen and modified from the Action Script. Whole list of mib parameters you could get by communicating with DMP via HTTP, by typing in the browser address line:

```
https://XXXXXXXX:7777/get\_param?p=\*. \*
```

where `XXXXXXXX` is IP address of your box.

You could look up for parameter `init.version`, which holds current version of the DMP, or you could know the date of the build running by looking at `init.build` parameter. Please, refer to DMP MIB data document for more information.

2.6.1. Getting mib parameter

```
public function mibGet(key:String):String;
//key - string, indicating mib parameter to get
```

Function `mibGet()` returns the value of the mib "key". String parameter `key` could be one of the mibs, like "init.version" or "init.build". In following example we demonstrate how to get current IP address (1) of the DMP.

```
import com.cisco.dms.MediaPlayer;
    var myMediaPlayer:MediaPlayer = new MediaPlayer();
    var myIP = myMediaPlayer.mibGet("init.IP");           //(1)
```

2.6.2. Setting mib parameter

```
public function mibSet(key:String, value:String);
//key – string, indicating mib parameter to set
//value – string, indicating value to set
```

Function `mibSet()` sets a mib "key" to the value specified. String parameter `key` could be one of the settable mibs. In following example we reboot DMP programmatically after 5 seconds (2) by setting "mng.reboot" (1) mib string to 1.

```
import com.cisco.dms.MediaPlayer;
var myMediaPlayer:MediaPlayer = new MediaPlayer();
var reboot = function() {
    myMediaPlayer.mibSet("mng.reboot", "1");           //(1)
}
setInterval(reboot, 5000);                             //(2)
```

2.7. Logging to Syslog on the DMP

```
public function log (msg:String);
public function logInfo (msg:String);
public function logDebug (msg:String);
public function logPoP (msg:String);
msg – message to log
```

There are four different APIs provided for logging with DMP. In order for these APIs to work, you must have configured syslog via the DMPDM interface first.

Function `log()` logs to syslog a message you provide as a parameter. Function `logInfo()` logs there a message with "INFO" tag, and `logDebug()` logs debug message to syslog. Function `logPoP()` logs a Proof of Play message to syslog.

2.8. Playing and stopping video on the DMP

```
public function playVideoToObject(url:String, loop:Boolean, obj:Object);

//url - string, containing link to the movie
//loop - indication of whether we want to play video in loop or not
//obj - flash object to play video to
public function stopVideo();
```

Function `stopVideo()` stops video playback and sets video plane to black. Function `playVideoToObject()` plays a video url to an object, which should be created on stage (look at 1.1 for more information and usage example.)

2.9. Subscribing to messages with the DMP

```
public function subscribe();
```

Function `subscribe()` allows to subscribe and receive messages from Flash application. The Messaging API allows external messages to be pushed to SWFs playing within system from an outside system. Messages are brokered through `sysmng`

and handed to Flsh Playback, which in turn fires events into any SWF that has subscribed to messages.

```
import com.cisco.dms.MediaPlayer;
var myMediaPlayer:MediaPlayer = new MediaPlayer();

var totalMessages:Number = 0;

var listenerObject:Object = new Object(); // (1)
listenerObject.messageHandler = function(msg:String)
{
    playlistBox.text = "Message: " + msg;
    totalMessages++;
    textBox.text = totalMessages;
}

myMediaPlayer.addListener(listenerObject); // (2)
myMediaPlayer.subscribe(); // (3)
```

At line (1) We create a listener for "messageHandler", then add listener to the MediaPlayer object (2), and subscribe for messages at line (3).

The example listens for an event of type "messageHandler". When a message of that type is received, it will be converted into a "messageHandler" event and dispatched.

Next to fire a message into the system using the DMP web interface:

- a) Navigate to Display Actions -> Flsh Playback on the navigation bar
- b) Fill in the Message Type as "messageHandler"
- c) Fill in the Message Payload as any text you would like to send to the SWF.
- d) Click "Send"

You will see your message appear on the screen and the message count at the bottom will be incremented.

```
public function unsubscribe();
Function unsubscribe(); allows to unsubscribe from Event Messages.
```

Handling Vertical Design

- a) If you use DMD (Digital Media Designer) for your presentation creation, please follow these steps:
 1. use portrait mode in DMD to create presentation
 2. if the video included into presentation, it needs to be rotated in video editing application and rendered to a supported format
 3. set the box rotation by using ROTATE MIB and save it print In browser window:

https://XXX.XX.XXX.XX:7777/set_param?init.ROTATE=90&mib.save=1

Where XXX.XX.XXX.XX is IP address of your box,
set_param?init.ROTATE=90 rotates the display orientation to 90 degree
&mib.save=1 saves your setting.
 4. Save settings and restart the DMP
- b) If you create standalone application, please follow these steps:
 1. design must be designed specifically sideways
 2. video needs to be rotated to 90° or 270°

Reference Information:DMP4310_API.docx and DMP4310_MIB.docx

NOTE: Cisco does not develop, maintain, sell, or support 3rd party tools and open source codes. Nor do we warrant that it is suitable for any purpose.