

Working With Files

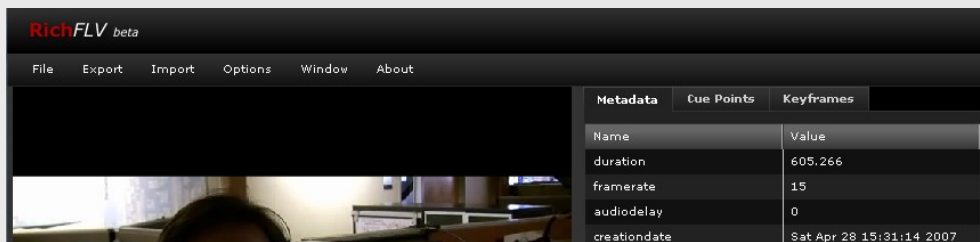


Agenda

- Introduction
- Samples / RichFLV
- The File API
- Registering File Types
- The Updater Class
- Working with Bytes

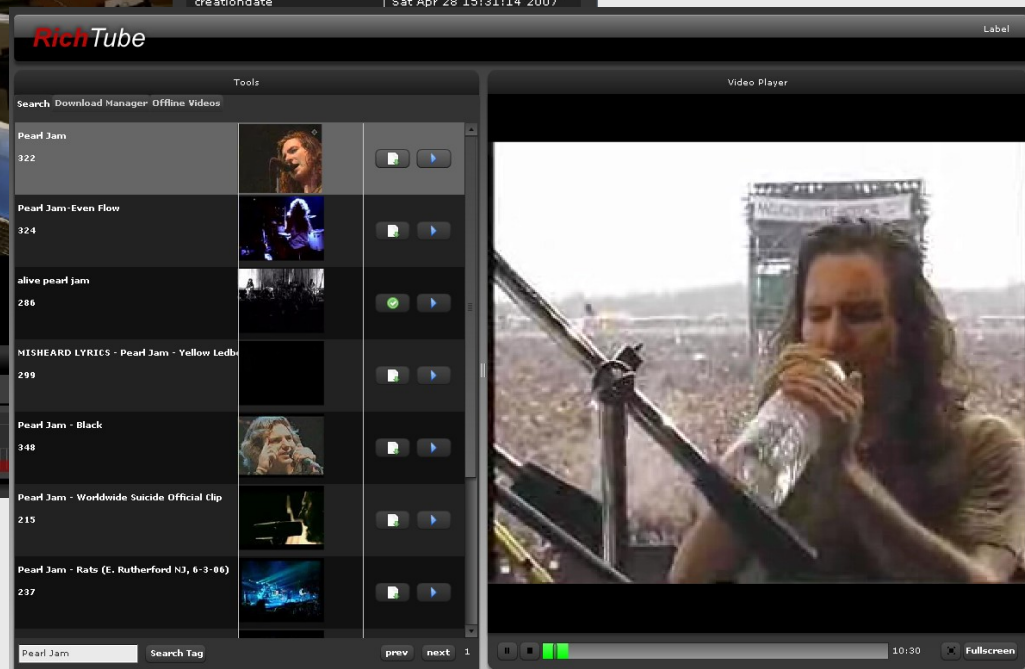
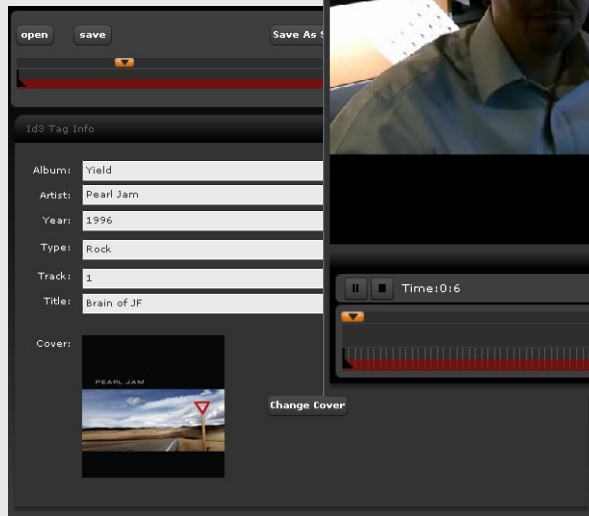
Some Samples

RichFLV



RichTube

RichMp3



File API

File API: The Possibilities

- Create And Delete Files/Directories
- Copy Files/Directories
- Move Files/Directories
- Get Information about Files and Directories
- Read and Write Files (binary/text)

File API: The Classes

- Package `flash.filesystem.*`
- Only 3 new classes do the Trick
 - `File` (represents a file or directory)
 - `FileStream` (methods for reading and writing to a file)
 - `FileMode` (sets the mode in which a file is accessed)

The Class `flash.filesystem.File`

- The class `File` is a pointer to a file
- So how do we point to a file
- Keep in mind that filepaths are platform dependent!

The `File` class has two properties which define a path

- `nativePath`
- `url`

The Class `flash.filesystem.File`

Using `nativePath`

```
var myFile:File=new File();  
myFile.nativePath="C:\\Users\\Apollo\\Desktop\\DMB - 27 - 08-01-07.mp3";
```

Using `url`

```
var myFile:File=new File();  
myFile.url="file:/C:/Users/Apollo/Desktop/DMB - 27 - 08-01-07.mp3";
```

The Class `flash.filesystem.File`

The File class has some static properties for system paths that abstract the differences Between the different Operating Systems.

- `File.appStorageDirectory`
- `File.appResourceDirectory`
- `File.currentDirectory`
- `File.desktopDirectory`
- `File.documentsDirectory`
- `File.userDirectory`

You Can use the resolve method to construct the path:

```
var file:File = File.desktopDirectory.resolve(„MyDirectory/file.txt“);
```

The Class `flash.filesystem.File`

URL Schemes

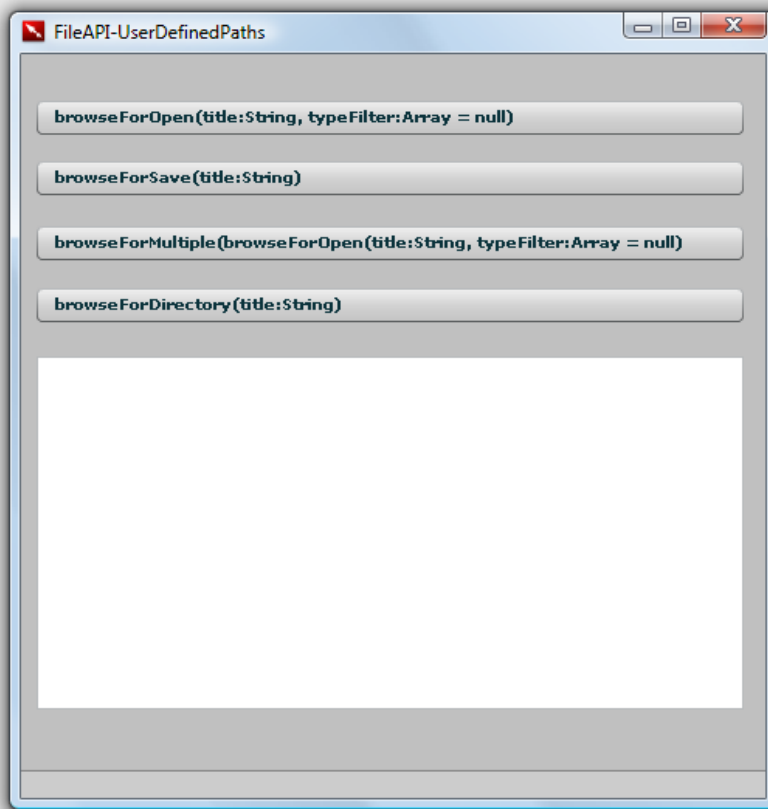
- file `file:///c:/test/myFile.txt`
- app-resource `app-resource:/test/myFile.txt`
- app-storage `app-storage:/test/myFile.txt`

The Class `flash.filesystem.File`

User defined paths

- `browseForOpen(title:String, typeFilter:Array = null)`
- `browseForSave(title:String)`
- `browseForMultiple(title:String)`
- `browseForDirectory(title:String)`

Example: User Defined Paths



The Class `flash.filesystem.File`

Retrieving information for files and directories

- `creationData`
- `creator`
- `exists`
- `extension`
- `icon`
- `isDirectory`
- `modificationDate`
- `name`
- `parent`
- `size`
- `type`
- `url`

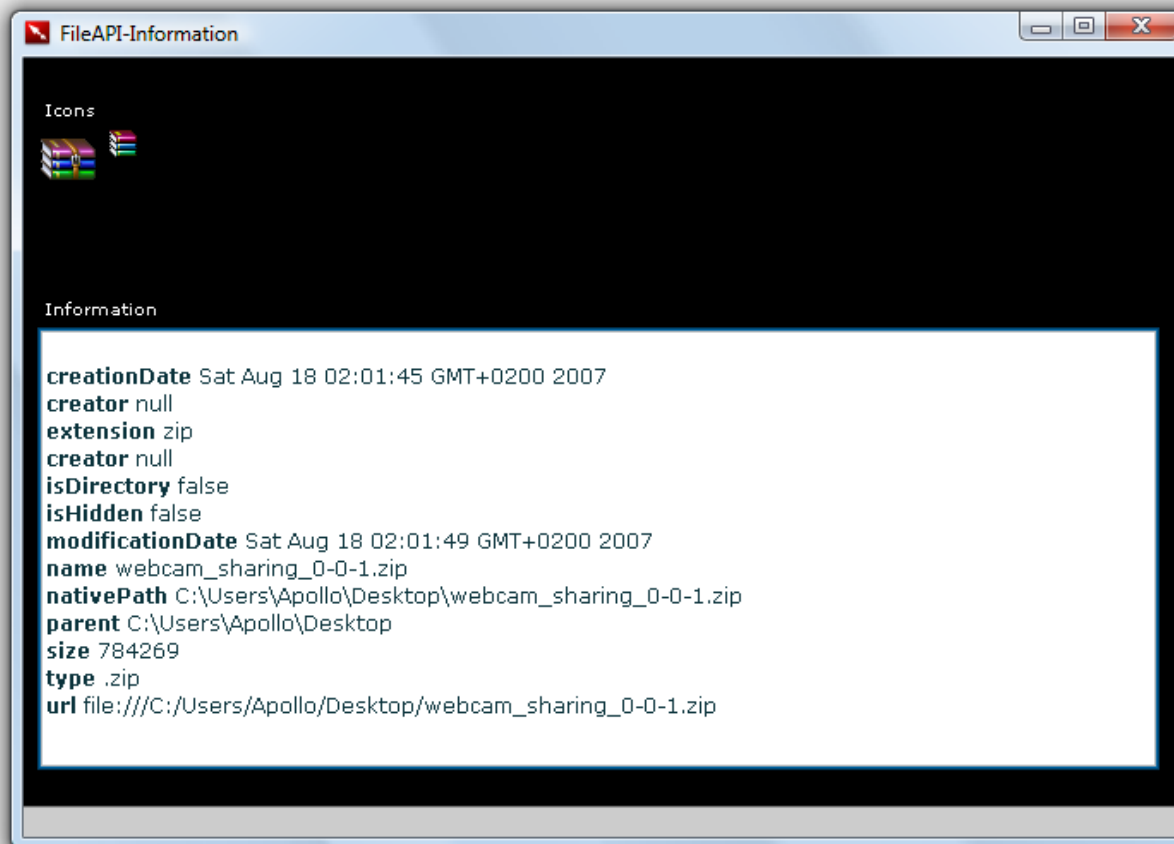
The Class `flash.filesystem.File`

Retrieving information for about the file system

- `File.encoding`
- `File.lineEnding`
- `File.seperator`
- `File.systemCharset`

- `Capabilities.os`

Example: Information about a file



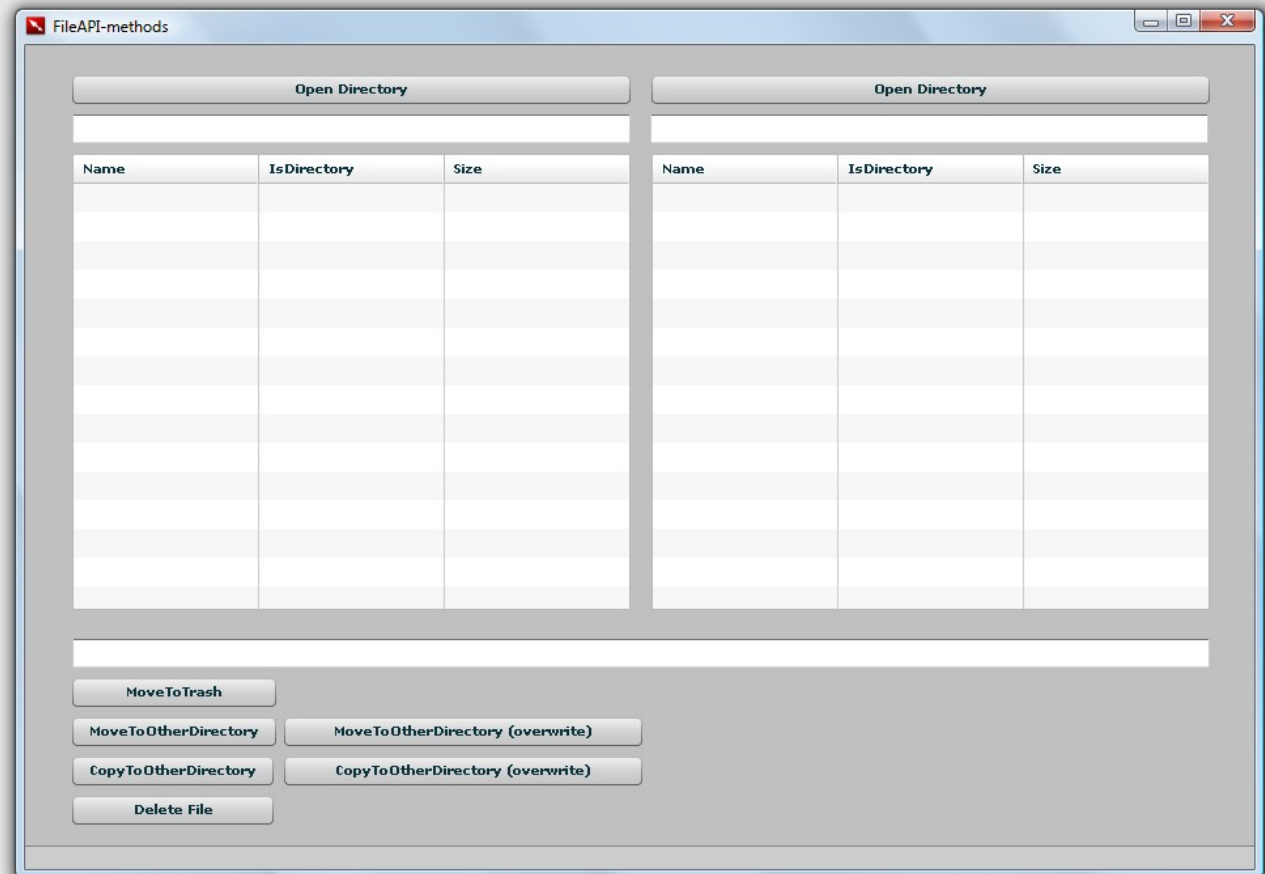
The Class `flash.filesystem.File`

Useful methods of the File API

```
createDirectory();  
createTempDirectory();  
listDirectory();  
copyTo()  
deleteDirectory()  
moveToTrash()
```

```
createTempFile();  
deleteFile();  
copyTo();  
moveTo()
```

Examples: File API Methods



FileStream

- package: flash.filesystem.*
- Lets you read and write to the file system
- Like ByteArray it implements IDataInput and IDataOutput

Example: Opening a file with read access

```
var file:myFile = File.desktopDirectory.resolve("myFile.txt");
```

```
var fileStream:FileStream = new FileStream();
```

```
fileStream.open(file, FileMode.READ);
```

FileMode

- Contains static constants that define the mode in which the file is opened
- You pass these constants to the `open()` or `openAsync()` methods of the `FileStream`

Possible modes are:

- | | |
|--------------------------|--|
| - FileMode.READ | read-only. File must already exist. |
| - FileMode.WRITE | write-only. Overwrites existing files ! |
| - FileMode.APPEND | write-only. Data can be written at the end |
| - FileMode.UPDATE | Data can be written everywhere |

Example: FileStream / FileMode



File API: Asynchronous/Synchronous methods

Synchronous

- blocks code execution until the file is completely read
- good for small files
- Easier to use / less code

Asynchronous

- runs in the background without blocking code execution
- informs about the end of the read operation through events
- good for large files

File API: Asynchronous/Synchronous methods

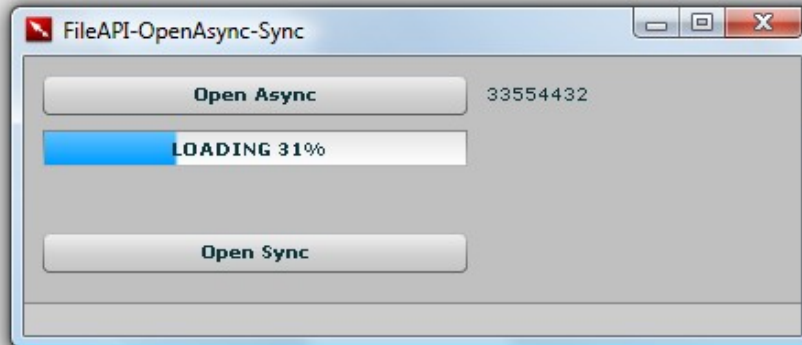
Synchronous

File.copyTo()
File.deleteDirectory()
File.deleteFile()
File.listDirectory()
File.moveTo()
File.moveToTrash()
FileStream.open()

Asynchronous

File.copyToAsync()
File.deleteDirectoryAsync()
File.deleteFileAsync()
File.listDirectoryAsync()
File.moveToAsync()
File.moveToTrashAsync()
FileStream.openAsync()

Example: Sync/Async methods



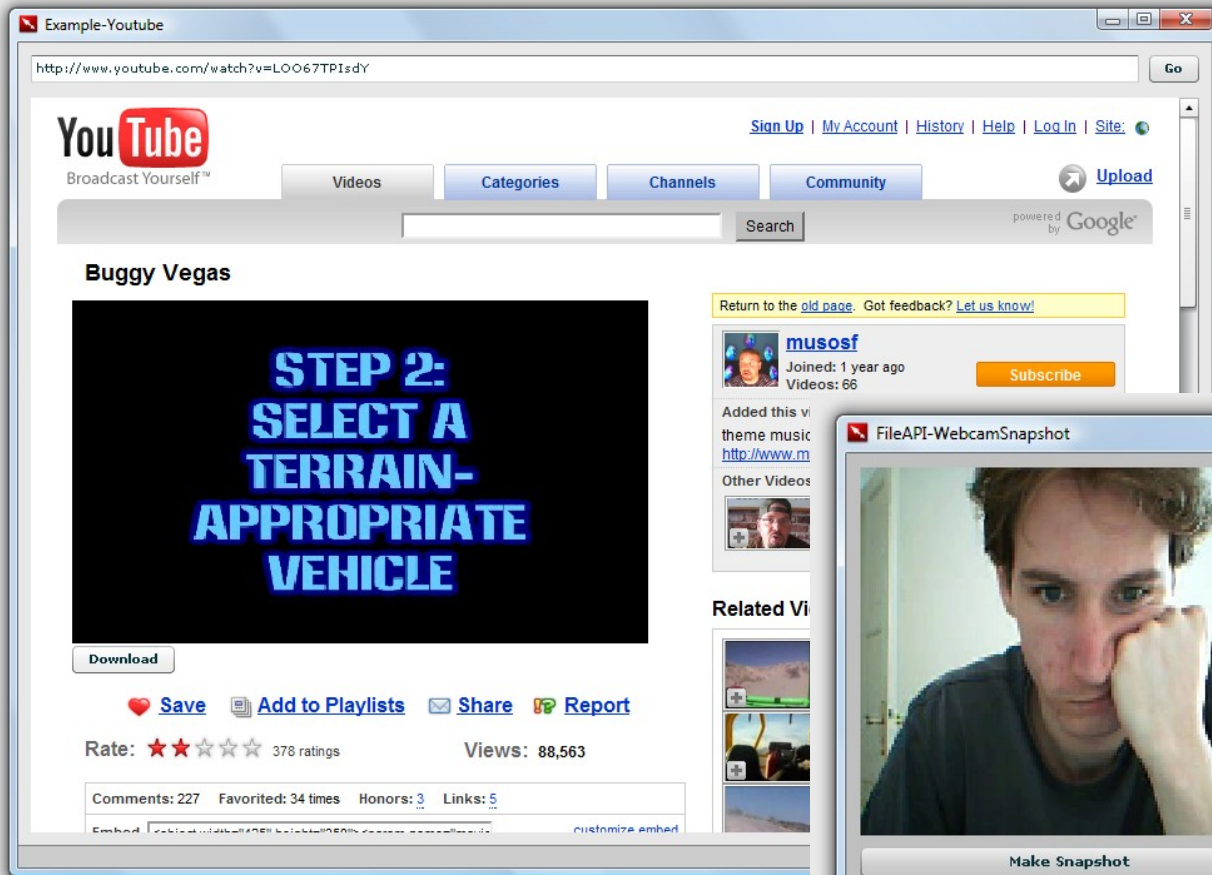
Registering File Types

```
<fileTypes>
  <fileType>
    <name>adobe.VideoFile</name>
    <extension>avf</extension>
    <description>Adobe Video File</description>
    <contentType>application/vnd.adobe.video-file</contentType>
  </fileType>
</fileTypes>
```

Example: File Serialization



Example: Putting it all together



The Updater class

```
var updater:Updater = new Updater();  
Updater.update(file:File, version:String);
```

- It's your responsibility to check for updates
- It's your responsibility to download the new .air file

Usual Update Scenario

Open Application Descriptor File and get version information



Load XML from Server and extract version information



Load new AIR file from Server / Save it to user harddisk

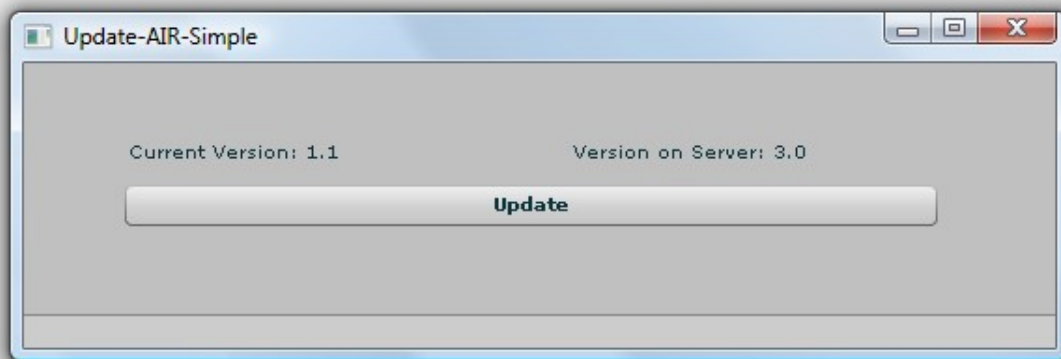


Update the Application with the downloaded file and the Updater Class

AIR Remote Updater

- Developed by Claus Wahlers
- <http://codeazur.com.br/lab/airremoteupdater/>
- Simplifies version checking
- No XML with version information needed
- Extracts AIR (zip) and reads version info
- Uses Fzip (also developed by Claus <http://codeazur.com.br/lab/fzip/>)

Example: Updating AIR Apps



Working With Bits And Bytes

Binary Data: Motivation

- Add new functionality independent of Flash Player releases
- All file formats have something in common. They are made out of Bits
- Implement any file format
- The only problem: Will you be able to show/play it?

Binary Data: Basics

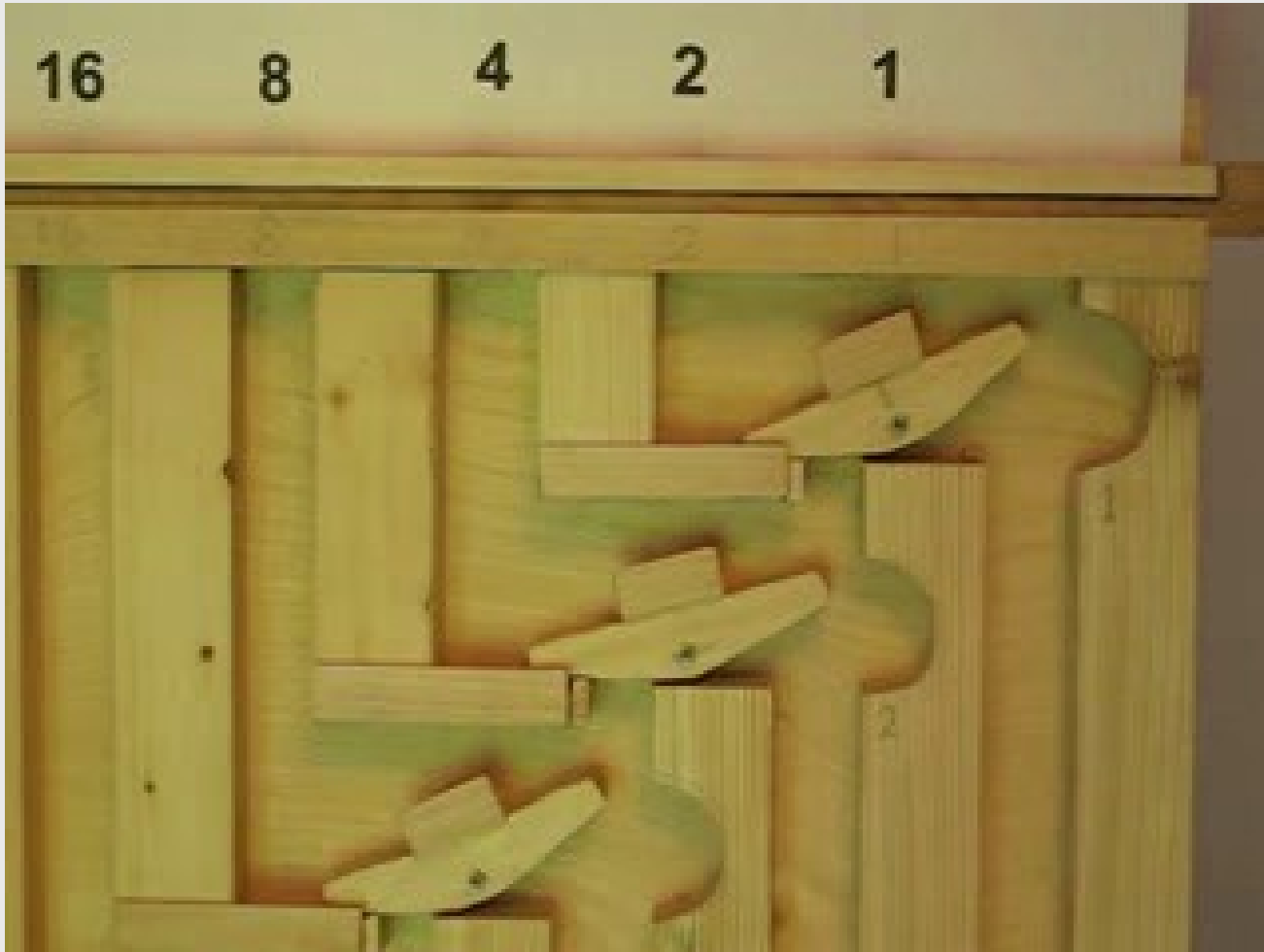
- Smallest information unit in a computer is a bit
- A bit can be 0 or 1
- Bits are often grouped together as Bytes (8 bits)
- Grouped bits are also called a “word”

Binary Data: Basics

- Counting with bits:

1	0000 0000
2	0000 0001
3	0000 0010
4	0000 0011
5	0000 0100
6	0000 0101
7	0000 0111
...	
255	1111 1111

Binary Data: Basics



<http://woodgears.ca/marbleadd/>

Binary Data: Basics

- Bit lengths of DataTypes in Actionscript

Number	8 Bytes
int	4 Bytes
uint	4 Bytes
String	variable (1 byte / char)

Binary Data: The Byte Array Class

- Available since Flash Player 9
- AIR adds two new methods: `deflate()` and `inflate()` for compression
- It's an array - so you can access bytes like this: `myByteArray[index]`
- If you read from a `ByteArray` the `position` attribute will be incremented

```
byteArray.position=0;
```

```
byteArray.readInt();
```

```
trace(byteArray.position) // outputs 32
```

Binary Data: The Byte Array Class

Methods for reading and writing bytes/special data formats:

`readByte()` – reads 8 Bits

`readInt()` - reads a 32 bit integer

`readShort()` – reads a 16 bit integer

`readUTF()` – reads a UTF string of variable length

and many more...

Binary Data: Operators – Bitwise AND

Bitwise AND

- Sign is & (compare logical AND &&)
- useful for analyzing packed words
- used to extract and test bits
- Rule: everything times zero is zero.
- Same as multiplication in the decimal world. $0 \text{ AND } 0 = 0$

-

$0 \text{ AND } 1 = 0$

$1 \text{ AND } 0 = 0$

$1 \text{ AND } 1 = 1$

You see that only if both Bits are set the result will be a set Bit.

	1	1	0	0	1	0	1	0
&	0	1	1	1	0	1	1	0
=	0	1	0	0	0	0	1	0

Binary Data: Operators – Bitwise AND

Bit Testing

- Create a mask with just the bits set that you want to test
- If the bits are set the result of the AND operation should be the same as the mask

Example:

Testing if the second least significant bit is set

$$\begin{array}{r} \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ \hline \end{array} \\ \& \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline \end{array} \\ \hline = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline \end{array} \end{array}$$

Binary Data: Operators – Bitwise AND

Bit Clearing

- Another useful scenario for using the bitwise AND is the so-called bit clearing.
- For any position in the bit array which you want to preserve you use 1
- and for any bit to clear you use 0 in your mask.

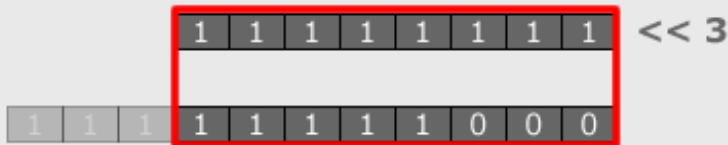
$$\begin{array}{r} \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ \hline \end{array} \\ \& \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline \end{array} \\ \\ = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ \hline \end{array} \end{array}$$

Binary Data: Left Shift (<<)

- (bit array) << (number of shifts)
- Shifts the bits to the left by the number of bits indicated by y.
- On the right side the bits are filled with zeros.
- On the left side they are simply cut off.

Example:

1111 1111 << 3 = 1111 1000

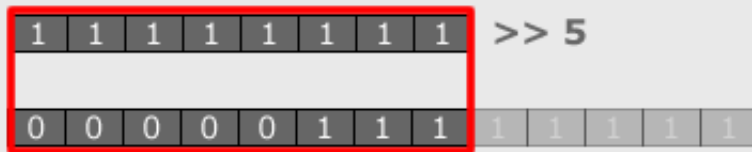


Binary Data: Right Shift (>>)

- Works like Left Shift operator but shifts the bits to the right.
- Bits on the left are filled with 0`s.
- Bits on the right are cut off.

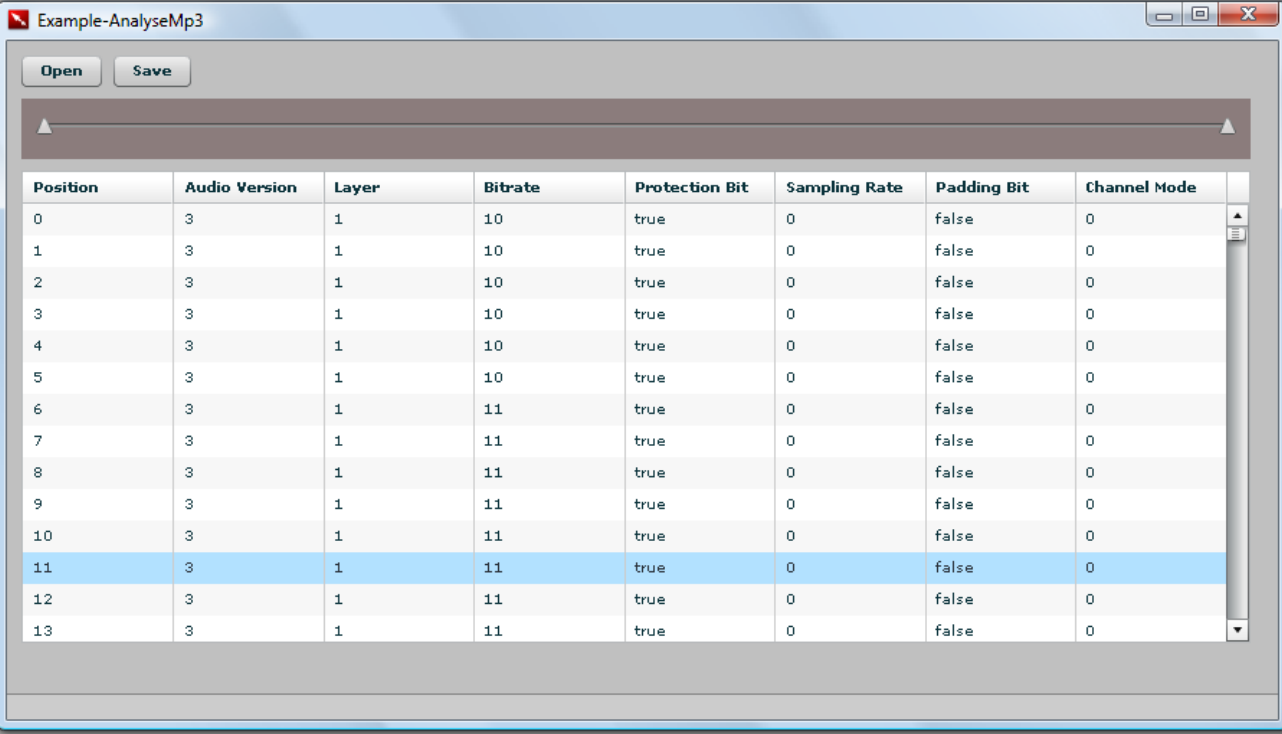
Example:

1111 1111 >> 3 = 0001 1111



Bits And Bytes Example

- Analyzing mp3 frames
- Cut mp3 files at frames



The screenshot shows a window titled "Example-AnalyseMp3" with "Open" and "Save" buttons. Below the buttons is a horizontal scrollbar. The main content is a table with 8 columns: Position, Audio Version, Layer, Bitrate, Protection Bit, Sampling Rate, Padding Bit, and Channel Mode. The table contains 14 rows of data, with the row at Position 11 highlighted in blue.

Position	Audio Version	Layer	Bitrate	Protection Bit	Sampling Rate	Padding Bit	Channel Mode
0	3	1	10	true	0	false	0
1	3	1	10	true	0	false	0
2	3	1	10	true	0	false	0
3	3	1	10	true	0	false	0
4	3	1	10	true	0	false	0
5	3	1	10	true	0	false	0
6	3	1	11	true	0	false	0
7	3	1	11	true	0	false	0
8	3	1	11	true	0	false	0
9	3	1	11	true	0	false	0
10	3	1	11	true	0	false	0
11	3	1	11	true	0	false	0
12	3	1	11	true	0	false	0
13	3	1	11	true	0	false	0

Mp3 File Format

What do we need to know if we want to cut an mp3 file?

- Something about the anatomy of the file
- The structure of the file
- Its specification

<http://mediasrv.ns.ac.yu/extra/fileformat/modules/mp3/mpeghdr.htm#MPEGTAG>

Mp3 File Format Structure

- Mp3 consists of independent frames
- A Mp3 Frame consists of a Header and the actual Data



Mp3 File Format

- Mp3 consists of independent frames
- A Mp3 Frame consists of a Header and the actual Data

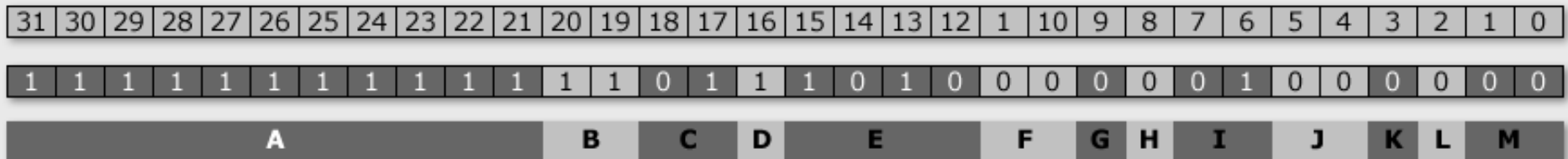


Mp3 File Format

So What Do We Need To Know About A Frame?



Mp3 Header



A: Sync Word

B: MPEG audio version (MPEG-1, 2, etc.)

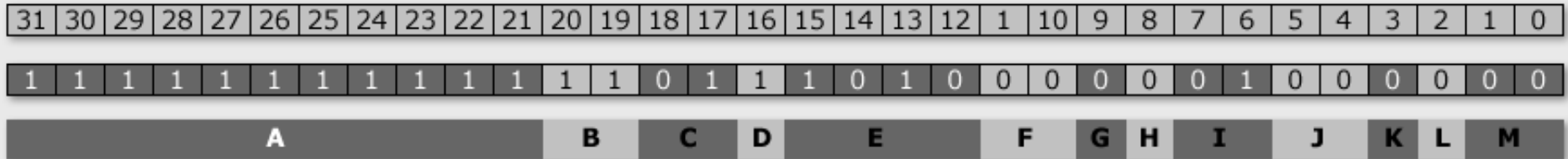
C: MPEG layer (Layer I, II, III, etc.)

D: Protection (if on, then checksum follows header)

E: Bitrate index (lookup)

F: Sampling rate frequency (44.1kHz, etc., determined by lookup table)

Mp3 Header



G: Padding bit (on or off, compensates for unfilled frames)

H: Private bit (on or off, allows for application-specific triggers)

I: Channel mode (stereo, joint stereo, dual channel, single channel)

J: Mode extension (used only with joint stereo, to conjoin channel data)

K: Copyright (on or off)

L: Original (off if copy of original, on if original)

M: Emphasis (respects emphasis bit in the original recording; now largely obsolete)

Finding Mp3 frames

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	1	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0

```

public function getNextFrame():Boolean
{
    while (file.bytesAvailable > 4) {
        var ch:int = file.readByte() & 0xff;
        if (ch != 0xff) {
            continue;
        }
        if ((file.readByte() & 0xe0) == 0xe0) {
            var filePos:int=file.position;
            file.position=filePos - 2;
            return true;
        }
    }
    return false;
}

```



	1	1	1	1	1	1	1
&	1	1	1	1	1	1	1
=	1	1	1	1	1	1	1

Finding Mp3 frames

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	1	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0



```
public function getNextFrame():Boolean
```

```
{
    while (file.bytesAvailable > 4) {
        var ch:int = file.readByte() & 0xff;
        if (ch != 0xff) {
            continue;
        }
        if ((file.readByte() & 0xe0) == 0xe0) {
            var filePos:int = file.position;
            file.position = filePos - 2;
            return true;
        }
    }
    return false;
}
```



	1	1	1	1	1	0	1	1
&	1	1	1	0	0	0	0	0
=	1	1	1	0	0	0	0	0

(0xe0)

Extracting the MPEG Audio ID

1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0



We want those Bits!

Extracting the MPEG Audio ID

>> 19

1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0					
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Extracting the MPEG Audio ID

	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
>> 19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
mask & 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Extracting the MPEG Audio ID

	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0		
>> 19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1		
mask & 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	
=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

The Next Steps – Grab some specifications!

The FLV Header

All FLV files begin with the following header:

FLV File Header

Field	Type	Comment
Signature	UI8	Signature byte always 'F' (0x46)
Signature	UI8	Signature byte always 'L' (0x4C)
Signature	UI8	Signature byte always 'V' (0x56)
Version	UI8	File version (for example, 0x01 for FLV version 1)
TypeFlagsReserved	UB[5]	Must be 0
TypeFlagsAudio	UB[1]	Audio tags are present
TypeFlagsReserved	UB[1]	Must be 0
TypeFlagsVideo	UB[1]	Video tags are present
DataOffset	UI32	Offset in bytes from start of file to start of body (that is, size of header)

<http://www.adobe.com/licensing/developer/>

Resources

Byte Arrays

- <http://www.bytearray.org>
- <http://www.richapps.de> ;-)
- <http://www.easycalculation.com/binary-converter.php>

File API

- http://labs.adobe.com/wiki/index.php/AIR:Articles:Apollo_Local_File_System
- <http://www.trajiklyhip.com/blog/index.cfm/2007/8/12/Working-With-the-File-System-API-in-AIR>
- <http://blog.kevinhoyt.org/2007/03/19/apollo-file-api-overview/>

MP3 Format

- <http://mediasrv.ns.ac.yu/extra/fileformat/modules/mp3/mpeghdr.htm#MPEGTAG>

FLV

- <http://www.adobe.com/licensing/developer/> (SWF & FLV Specification)

Thanks for listening!

Questions & Comments: benz@richapps.de