Net-X-Player

API

www.net-x-code.com

Version 5.5.597 (2018-03-28)

Table of Contents

Overview

Net-X-Player is an HTML 5 browser video player optimized for post production use. It's main features include:

- Frame accurate seeking
- Time code support
- Extended playback controls
- Full keyboard control
- Contour and other external Jog/Shuttle support
- Custom event generation

There are two ways of using Net-X-Player.

The first is as a self-contained player that uses its own internal visual controls as-is. The main benefit of this mode is that the player is extremely easy to use and the HTML elements necessary to implement the player are kept to a bare minimum. Please see the file index_embed.html for an example. This is the easiest way to use Net-X-Player.

The second is as a controller to which you can attach a user-defined control interface. This way you could create a completely custom visual interface and use Net-X-Player for its control functions only. The file “videoplayer.html” shows how such a custom interface might be built. Although this interface is more flexible, it does demand a more intimate knowledge of HTML and Javascript as well as the API calls defined in this document.

Both of these methods use the various API calls described in this document. Not all API calls are used by both interfaces. Some API calls are not used by either of them but are there for added functionality that could be called by the user should they be needed for maximum user control and flexibility of operation.

For the quickest and most effective introduction to Net-X-Player, we suggest using Net-X-Player as a self-contained player first. Have a look at the way the video container is defined in the index_embed.HTML file and the way that the individual video files (both rtin and non-rtin) are defined and linked. Once you are comfortable with using Net-X-Player in that fashion, you can extend your control over the interface by creating your own interface and wiring up your own visual elements to the Net-X-Player API calls.

As with all software and all documentation about that software, there are many additions and modifications over time. Please use the most up to date versions of both and please let us know how we can improve.

Thank you for using Net-X-Player.

Quick Start:

Using Net-X-Player and its self-contained controls as a player

1) Look at index_embed.html.
2) Find the line that sets the src=”media/LBR.mp4". Change that line to the video file that you wish to use as the default. If you have an RTIN file for that video, give it the same name with an .rtin extension and store it in the same subdirectory/folder.
3) Look at the <table> section toward the end of the document. Add/modify the thumbnail images and/or the video files used to suit your needs.
4) Skip the rest of this document.

*(Yes, we know that using “on-click()” in the HTML is not considered unobtrusive javascript but it greatly simplifies understanding the example)

Setup and Flags

To add a Net-X-Player to your web page you must include the supporting css and js files within the head tag.

```
<head>
        <!-- drastic css is required to place the Net-X-Player elements -->
        <link href="css/video-dt.css" rel="stylesheet" type="text/css">
        <!-- netxplayer.js must be in the <head> for older IEs to work. -->
        <script src="js/netxplayer.js"></script>
</head>
```

These must go in the head, especially for Internet Explorer.

To add the player in your page, add the following HTML:

```
  <!-- Begin Net-X-Player -->
  <div class="video-dt-box" id="dl-video-dt-box">
    <video id="dl-videoPlayer" class="video-js" width="640" height="360" playsinline webkit-playsinline preload poster="images/drastic.png">
      <source id="dl-mp4video" src="media/15 Min Timecode Pal NTSC Frames - Videoamt.mp4" type='video/mp4;' />
      <!-- Flash Fallback. Use any flash video player here. Make sure to keep the vjs-flash-fallback class. -->
      <object id="flash_fallback_1" class="vjs-flash-fallback" width="640" height="360" type="application/x-shockwave-flash"
        data="js/video-js.swf">
        <param name="movie" value="js/video-js.swf" />
        <param name="allowfullscreen" value="true" />
        <param name="flashvars"
          value='config={"playlist":["images/drastic.png", {"url": "15 Min Timecode Pal NTSC Frames - Videoamt.mp4","autoPlay":false,"autoBuffering":true}]}' />
        <!-- Image Fallback. Typically the same as the poster image. -->
        <img src="images/drastic.png" width="640" height="360" alt="Poster Image"
          title="No video playback capabilities." />
      </object>
    </video>
  </div>
  <!-- End NetXPlayer -->
```

Finally, at the very bottom of the file, just above the </body> tag, the player initialize function must be called.

```
<script>
  window.onload = dl.initializePlayer();
</script>
```

IMPORTANT: Each video player instance is linked to a user-defined variable. That variable needs to be correctly placed in a few different places in your HTML page. For the purposes of this documentation we have used a two character variable named “dl”. The reason to have this variable is to allow multiple videos on the same page (See the index_2video.html file for an example of having 2 videos on the same HTML page)

1) As the first 2 characters of the “id” fields for 3 HTML lines:

```
<div class="video-dt-box" id="dl-video-dt-box">
<video id="dl-videoPlayer" class="video-js" width="640" height="360"  ...
<source id="dl-mp4video" src="media/LBR_000.mp4" type='video/mp4;' />
```

Please note that all 3 “id” fields in the above 3 lines start with “dl-”. (You may, of course, change that to another name but it must be consistently applied as per this documentation) Also note that although the variable name is “dl” that there is also the requirement of a hyphen following the variable name FOR THIS STEP ONLY.

2) In the <script> section where you see the dl variable linked to the Net-X-Player definition:
var dl = NetXPlayer({ …
and
instance : 'dl',

3) In the onload section:
window.onload = dl.initializePlayer();

That's it!

On the <video> line there are optional flags to allow more user control:

- controls – enable 'in video' controls
- nocontrols – disable under video controls
- noclip – disable the “clip” option in the video controls group
- preload – preload the video, if possible
- autoplay – start playing on load

- loop – loop playback at the end

MP4, RTIN and Live RTIN

The Net-X-Player generally uses a browser native decode for the media (mostly MP4) files. If there is an RTIN available, it will be next to the media file, with the same name and the RTIN extension. These are picked up automatically by the HTML5 layer of the player, and the per frame time code is used automatically with the source file.

Another use case of the Net-X-Player is to play unsupported HTML5 video formats (MXF, MOV, AVI, etc.) that have supported essence and to play supported formats that are currently open for recording, that cannot be opened by the native decoder. In this case, Net-X-Player uses a custom HTML5 or Adobe Flash component to open and read the RTIN directly, avoiding reading the unclosed/open MP4 file. For this to work, the server must allow progressive HTTP requests on port 80.

Optional Flash Player (not recommended for new installs)

The Flash version of the player is no longer recommended, but if using Flash, it must be enabled in the browser you are using and to allow the server to receive the requests, a Flash socket server must be run on the HTTP server, so that Flash will not block the partial requests to the HTTP server. This is a security check in flash, and cannot be overridden any other way.

The easiest way to achieve this is to use an XML file and a PHP program. The XML file should look like this:

```
<cross-domain-policy>
    <allow-access-from domain="*" to-ports="80" />
</cross-domain-policy>
```

The PHP program is available at the link above, and is listed in the appendix to this document as well.

Once the above are set up, the RTIN can be specified in the video tag just like the MP4 would be. The only other thing to add is to make the flash technology the default choice for playback, with data-setup='{“techOrder”:[“flash”]{' like the example below:

```
   <video id="Video1" class="video-js vjs-default-skin" width="640" height="264" controls data-setup='{"techOrder":["flash"]}'>
      <source src="media/LBR_000.rtin" type='video/mp4;' />
      <!-- Flash Fallback. Use any flash video player here. Make sure to keep the ntx-flash-fallback class. -->
      <object id="flash_fallback_1" class="nxc-flash-fallback" width="640" height="264" type="application/x-shockwave-flash"
```

```
        data="js/video-js.swf">
        <param name="movie" value="js/net-x-player.swf" />
        <param name="allowfullscreen" value="true" />
        <param name="flashvars"
          value='config={"playlist":[{"url":
"media/LBR_000.rtin","autoPlay":false,"autoBuffering":false}]}' />
      </object>
    </video>
```

Functions

Note: a trailing “*” in the function name indicates that function has not been created and tested yet.

Playback Control

Basic transport control and status for playback

setPlayMode(nPlayMode)*

getPlayMode()*

Returns the current play mode (1=Normal, 2=Loop)

play()

Play forward at normal play speed (1x). Returns void.

shuttle(speed)

NOTE: same as playAtSpeed()
1.0 is regular speed forward, -1.0 is regular speed backward, 2.0 is forward twice as fast, etc. Returns void.

shuttleOffset(increment)

Increment (decrement if negative) the current speed/direction by an incremental amount. Returns void.

pause()

Pause playback at the current location. Returns void.

paused()

Returns true if paused and false if not.

playPause()

Toggles between Play and Pause for the running video. Returns void. (*** only called in keyboard functions!)

fastPlayParams(fastIntervalMs, fastSpeedFactor, fUsePlayInFFwd, fUsePlayInFRwd)

Set the time interval in milliseconds for the jumps when doing fast forward or rewind. If just fastIntervalMs is set, then fastSpeedFactor will be (1000 / fastIntervalMs) and fUsePlayInFFwd and fUsePlayInFRwd will be false. Return void.

- fastIntervalMs – time between seeks in milliseconds

- fastSpeedFactor – multiplier for speed to match the fastIntervalMs
- fUsePlayInFFwd – play the stream in play during the fast reverse seeks (default false)
- fUsePlayInFRwd – play the stream in play during the fast forward seeks (default false)

getMaxFastPlaySpeed()

Get the maximum allowable fast forward or reverse speed (default is 10x). Returns Number.

setMaxFastPlaySpeed(maxSpeed)

Set the maximum allowable fast forward or reverse speed. No larger than 10x is recommended, as it will use seeks to emulate speeds above 10x. Returns void.

fastForward(speed)

Play fast forward (default is 5x). Returns void.

fastRewind(speed)

Play fast backwards (default 5x). Returns void.

state()

Returns: Fast Reverse, Rewind, Shuttle Backward, Pause, Shuttle Forward, Play, or Fast Forward depending on direction and speed of playback. Returns String.

seekRelative(nFramesDistance)

Seek a number of frame forward, if positive, or backwards, if negative. Returns Void.

seek(location, nType)

Seek to a specific frame, second or time code depending on nType. For more information on nType, see the Duration() function. Returns Void.

seekForward(stepFrames)

NOTE: could be part of seek() (frames)
Seek forward from the current position stepFrames in frames. Returns Void.

seekBackward(stepFrames)

NOTE: could be part of seek() (frames)
Seek backward from the current position stepFrames in frames. Returns Void.

skip(seconds)

Seek in either direction (negative backwards/positive forwards) a number of seconds. Returns Void.
NOTE: could be part of seek() (seconds)

setLiveFollowSeconds(seconds)

Set the number of seconds back from the live recording head that we will play from.

getLiveFollowSeconds()

Get the number of seconds back from the live recording head that we will play from.

seekEnd()

Seek to the end of the file. If it is a live recording file, go into play and chase the record head. Returns Void.

offsetSecond (seconds)

Move a negative or positive offset from the current position in seconds. Returns Void.

ToAbsTimeCode() *

Not called by any function. (*** should it be documented?)

toTimeCode() *

Not called by any function. (*** should it be documented?)

stop()

Stop playback and return to the first frame of the file. Returns Void.

Status

duration(nType)

Returns the total duration of the file. The nType specifies the format to be returned:

- 1 – video frames (integer)
- 2 – seconds (float)
- 3 – time code (string)

lastFrame(nType)

Returns the last frame that can be displayed. This is normally one frame less than the duration/out point, but can be earlier.

- 1 = in video frames, returns Number
- 2 = in seconds, returns Float
- 3 = in timecode, return String

direction()

Return the current direction of the player: 0=stop/pause, -1=reverse, 1=forward. Returns Number.

curSpeed()

Returns the current speed the player is playing at as a float, where 1.0 is normal play speed. Returns Float.

curPosition(nType)

Return the current position within the file. See Duration() for the nType setting.

- 1 = in video frames, returns Number
- 2 = in seconds, returns Float
- 3 = in time code, return String

curTCType()

Returns the current time code type being used. Returns Number.

- 1 – 24FPS/Film
- 2 – NDF (Non Drop Frame)
- 4 – DF (Drop Frame)
- 8 – 25FPS/PAL
- 16 – 50FPS
- 32 – 59.94FPS DF
- 64 – 60FPS NDF

setTCType(nTcType)

Set the timecode type. Returns Void.

curTC()

Returns the current timecode as SMPTE. Returns String.

curUB()

Returns the current userbits. Returns String.

curUUID()

Returns the current UUID for the file, or “” if there is no UUID. It is returned in the form “8c11f9088d6411e58994feff819cdc9f”. Returns String.

curFrameType()

Returns I, B or P for the current frame (or “0” if not I, B, or P). Returns String.

curFPS()

Returns a float number with two decimal places, like 29.97 or 59.94.

getIsLive()

Returns true if the file loaded is currently recording, otherwise it returns false.

Audio

audioMute()

audioMute(bool)

If no variable is passed, toggle the audio mute (if muted, unmute and visa versa). If a bool (true/false) is supplied, then explicity mute or unmute the audio. Returns Void.

audioMuted()

Returns the current audio mute state. If true, then the audio is muted, else it will return false. Returns bool.

audioLevel(nLevel)

Set the audio level (min = 0.0, max = 1.0). Returns Void.

audioLevelSlider()

Sets the volume based on a slider in the HTML code with an ID of “audioslider”. Returns Void.

audioSetChannel(pairToSet)

Sets the pair of audio AAC channel to playback. Returns Void.

audioGetChannel ()

Get the current pair of audio AAC channel that is playing back. Returns Number.

audioGetMaxChannel()

Returns the maximum number of channel pairs in the current RTIN file. Returns Number.

Video

getWidth()*

getHeight()*

Returns the width or height of the video

getFrameRate()

Returns the frame rate of the file being played. Returns Number. (*** same as curFps())

curFileName()

Returns the name of the file being played. Returns String.

loadFile(”videofile”, cb)

Load but don't play the video file that is passed in. Set up in the HTML file (e.g. show thumbnail files and the user loads them by clicking on them. See index.html for an example). Note that you can pass a callback function as the (optional) second parameter. Returns Void.

In and Out

markIn()

Make the current time the in time. Returns Void.

setIn(szTimeCode)

Set the in time explicitly. Returns Void.

setInUb(szUB)

Set the in user bits explicitly. Returns Void.

getIn()

Return the current in point time code string. Returns String.

getinUb()

Return the current in point user bit string. Returns String.

seekIn()

Go to the current in point. Returns Void.

markOut()

Make the current time the out time. Returns Void.

setOut(szTimeCode)

Set the out time explicitly. Returns Void.

getOut()

Return the current out point time code string. Returns String.

getoutUb()

Return the current out point user bit string. Returns String.

seekOut()

Go to the current “out” time. Returns void.

updateIn()

When using the default GUI, this sets szinText = intcText.value. Returns Void.

setOut(szTimeCode)

Set the out time explicitly. Returns Void.

setOutUb(szUB)

Set the out user bits explicitly. Returns Void.

updateOut()

When using the default GUI, this sets szoutText = outtcText.value. Returns Void.

Events

eventOne()*

eventTwo()*

eventThree()*

Other

VersionString()

Returns the version of Net-X-Player that is running and also prints it on the console. e.g. "5.0.421"

dbgUrl()

Prints to the console the video file being loaded/played. Returns String. This is used during debugging and should not be very useful otherwise (*** document this?)

IsIE()

Returns true if the browser is IE.

IsEdge()

Returns true if the browser is Edge.

IsFF()

Returns true if the browser is FireFox.

querystring(key)

return the value of the variable passed on the URL line. e.g. if the URL is http://www.drastic.tv/netxplayer/index_embed.html?this=that then qurystring(this) would return “that”.

initializePlayer([szFileName | Null], [onLoadedFcn | Null])

Initializes and sets up the Net-X-Player control. The first parameter is an optional file name and path. If this is not used, the one in the HTML tag will be used. If can also be set to null, if the onLoadedFcn is required. The onLoadedFcn is a callback function that will be called when the file is initially loaded, and its metadata is valid. Returns Void. E.G.

```
function OnLoaded() {
 console.log('OnLoaded() duration = ' + dl.duration(1));
};
window.onload = dl.initializePlayer(szProxyName, onLoaded);
```

LoadFileRtin()

Load a .rtin file held in the variable szRTINFile or composed of the video’s root file name with an added .rtin extension. (*** should this be documented? It is only used by initializePlayer(), curUB(), and loadFile())

loadPlay(file, cb)

Loads and starts playing the named file and then calls the callback function (cb) if one was passed. Returns void.

setMakeClip(newMakeClipFcn)

Set the function to call when the make clip button is pressed. This is user-defined as this function does not come with Net-X-Player. Returns Void.

MakeLimit()

Function that only allows viewing the video between the Mark In time and the Mark Out Time. Returns void.

makeClip()

Calls via http a function to create the clip. Returns Void.

html5Controls(fActive)

Enable or disable the HTML5 controls within the player. If called without fActive, it will return the current state of the HTML5 controls.

requestFullscreen()

Request that the browser player goes full screen. This may be denied by the user's security settings, and can be checked with isFullscreen().

isFullscreen()

Returns a bool, true if the player is in fullscreen, otherwise false.

enableQtKeyboard()

Enable keyboard control of the player. Returns Number.

disableQtKeyboard()

Disable keyboard control of the player. Returns Number.

getGopAlignment()

Get the current type of GOP alignment in use. Returns Number.

0 – No alignment/frame accurate
1 – Align mark in and mark outs
2 – Align mark in, mark out and seeks
3 – Align mark in, mark out, seeks and pauses

setGopAlignment(nGopAlignType)

Set the current type of GOP alignment. See getGopAlignment() for possible settings. Returns Void.

gopFindPrevIFrame(frame)

Find closest Iframe before the passed in frame number.

gopFindNextIFrame(frame)

Find closest Iframe after the passed in frame number.

findClosestIFrame(frame)

Find closest Iframe to the passed in frame number. Calls gopFindPrevIFrame() and gopFindNextIFRame().

CurTCType()

Returns the current time code type being used for time code display and processing

- 1 – 24FPS/Film
- 2 – NDF (Non Drop Frame)
- 4 – DF (Drop Frame)
- 8 – 25FPS/PAL
- 16 – 50FPS
- 32 – 59.94FPS DF
- 64 – 60FPS NDF

SetTCType(nTcType)

Sets the current timecode type. See CurTCType above for settings.

videof functions (dl.videof.xxxxx())

videof.toTime(frame)

Returns the current timecode of the frame in the video in HH:MM:SS format.

videof.toSMPTE(frame)

Returns a SMPTE timecode in HH:MM:SS;FF format.

videof.toSeconds(SMPTE)

Converts a SMPTE timecode to Seconds.

videof.toMilliseconds(SMPTE)

Converts a SMPTE timecode to Milliseconds.

videof.toFrames(SMPTE)

Converts a SMPTE timecode to its equivalent frame number.

Appendix 1: Flash Policy Server PHP (legacy)

```
#!/usr/bin/env python
#
# flashpolicyd.py
# Simple socket policy file server for Flash
#
# Usage: flashpolicyd.py [--ipv6=1] [--port=N] --file=FILE
#
# Logs to stderr
# Requires Python 2.5 or later

from __future__ import with_statement

import sys
import optparse
import socket
import thread
import exceptions
import contextlib

VERSION = 0.1

class policy_server(object):
    def __init__(self, ipv6, port, path):
        self.ipv6 = ipv6
        self.port = port
        self.path = path
        self.policy = self.read_policy(path)
        self.log('Listening on port %d\n' % port)
        if ipv6 == 1:
                            try:
                                    self.sock = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
                            except AttributeError:
                                    # AttributeError catches Python built without IPv6
                                    self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                            except socket.error:
                                    # socket.error catches OS with IPv6 disabled
                                    self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        else:
                            self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.sock.bind(('', port))
        self.sock.listen(5)
    def read_policy(self, path):
        with file(path, 'rb') as f:
            policy = f.read(10001)
            if len(policy) > 10000:
                raise exceptions.RuntimeError('File probably too large to be a policy file',
                                              path)
            if 'cross-domain-policy' not in policy:
                raise exceptions.RuntimeError('Not a valid policy file',
```

```
                                    path)
        return policy
    def run(self):
        try:
            while True:
                thread.start_new_thread(self.handle, self.sock.accept())
        except socket.error, e:
            self.log('Error accepting connection: %s' % (e[1],))
    def handle(self, conn, addr):
        addrstr = '%s:%s' % (addr[0],addr[1])
        try:
            self.log('Connection from %s' % (addrstr,))
            with contextlib.closing(conn):
                # It's possible that we won't get the entire request in
                # a single recv, but very unlikely.
                request = conn.recv(1024).strip()
                if request != '<policy-file-request/>\0':
                    self.log('Unrecognized request from %s: %s' % (addrstr, request))
                    return
                self.log('Valid request received from %s' % (addrstr,))
                conn.sendall(self.policy)
                self.log('Sent policy file to %s' % (addrstr,))
        except socket.error, e:
            self.log('Error handling connection from %s: %s' % (addrstr, e[1]))
        except Exception, e:
            self.log('Error handling connection from %s: %s' % (addrstr, e[1]))
    def log(self, str):
        print >>sys.stderr, str

def main():
    parser = optparse.OptionParser(usage = '%prog [--ipv6=IPV6] [--port=PORT] --file=FILE',
                                   version='%prog ' + str(VERSION))
    parser.add_option('-i', '--ipv6', dest='ipv6', type=int, default=0,
                      help='listen on ipv6 instead of ipv4', metavar='IPV6')
    parser.add_option('-p', '--port', dest='port', type=int, default=843,
                      help='listen on port PORT', metavar='PORT')
    parser.add_option('-f', '--file', dest='path',
                      help='server policy file FILE', metavar='FILE')
    opts, args = parser.parse_args()
    if args:
        parser.error('No arguments are needed. See help.')
    if not opts.path:
        parser.error('File must be specified. See help.')

    try:
        policy_server(opts.ipv6, opts.port, opts.path).run()
    except Exception, e:
        print >> sys.stderr, e
        sys.exit(1)
    except KeyboardInterrupt:
        pass

if __name__ == '__main__':
    main()
```

Appendix 2: Flash Policy XML (legacy)

```
<cross-domain-policy>
    <allow-access-from domain="*" to-ports="80" />
</cross-domain-policy>
```

Appendix 3: Cross Domain Access

If you need to serve your HTML pages from one server, and your MP4/RTIN content from another, you will need to enable cross domain access. For security reasons, browsers restrict cross-origin HTTP requests initiated from within scripts. For example, XMLHttpRequest follows the same-origin policy. So, a web application using XMLHttpRequest could only make HTTP requests to its own domain. To allow the Net-X-Player to make requests to a different server, that server's configuration must be modified to allow the request. The following is a description of a setup to allow an Apache HTTP server to allow request from ANYWHERE in the network. It should not be used on production or internet facing systems. For those, the allowed source servers MUST be limited, as described below.

To allow requests from ANY server for development, add the following lines to your httpd.conf under the <VirtualHost section that has the files you need access to:

```
# Always set these headers.
Header always set Access-Control-Allow-Origin "*"
Header always set Access-Control-Allow-Methods "POST, GET, OPTIONS, DELETE, PUT"
Header always set Access-Control-Max-Age "1000"
Header always set Access-Control-Allow-Headers "x-requested-with, Content-Type, origin, authorization, accept, client-security-token"

# Added a rewrite to respond with a 200 SUCCESS on every OPTIONS request.
RewriteEngine On
RewriteCond %{REQUEST_METHOD} OPTIONS
RewriteRule ^(.*)$ $1 [R=200,L]
```

These lines basically change the headers being returned from your server, indicating to the caller that it can make cross domain requests to this server. As such, this change must be made on the server that is supplying the MP4 and RTIN files. It does not have to be made on the server supplying the HTML/CSS/etc. The reason for the second, rewrite, section is to make sure any 'preflight' requests (empty request to check availability) also work, just as the eventual request for the resource with.

There should also be a crossdomain.xml file in the root for the flash player in the browser to allow access to the RTIN file. Like the cross domain access for Apache, this file should be locked down to only allow connections from well known servers under your control. The file should look like this:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
        <allow-access-from domain="*"/>
</cross-domain-policy>
```

IMPORTANT:
The configuration above is for development and in house use only. To deploy to production servers, the <Header always set Access-Control-Allow-Origin "*"> line must be changed to only allow requests from the known servers that you expect to make those request. For example:

```
Header always set Access-Control-Allow-Origin "http://www.espn.com"
Header always set Access-Control-Allow-Origin "http://www.espn2.com"
Header always set Access-Control-Allow-Origin "http://199.181.132.250"
```

and the crossdomain.xml in the root of the file server should also be limited to the same sites.