Unofficial AirPlay Protocol Specification

- 1. Introduction
- 2. Service Discovery
  - 2.1. AirTunes service
  - 2.2. AirPlay Service
- 3. Photos
  - 3.1. HTTP requests
  - 3.2. Events
  - 3.3. Photo Caching
  - 3.4. Slideshows
- 4. Video
  - 4.1. HTTP requests
  - 4.2. Events
- 5. Audio
  - 5.1. RTSP requests
  - 5.2. RTP Streams
  - 5.3. Volume Control
  - 5.4. Metadata
  - 5.5. AirPort Express Authentication
  - 5.6. Remote Control
- 6. Screen Mirroring
  - 6.1. HTTP requests
  - 6.2. Stream Packets
  - 6.3. Time Synchronization
- 7. Password Protection
- 8. History
- 9. Resources
  - 9.1. IETF RFCs
  - 9.2. IETF drafts
  - 9.3. Apple Protocols

1. Introduction

AirPlay is a family of protocols implemented by Apple to view various types of media content on the **Apple TV** from any iOS device or iTunes. In this documentation, "**iOS device**" refers to an iPhone, iPod touch or iPad. The following scenarios are supported by AirPlay:

- Display photos and slideshows from an iOS device.
- Stream audio from an iOS device or iTunes.
- Display videos from an iOS device or iTunes.
- Show the screen content from an iOS device or OS X Mountain Lion. This is called **AirPlay Mirroring**. It requires hardware capable of encoding live video without taking too much CPU, so it is only available on iPhone 4S, iPad 2, the new iPad, and Macs with Sandy Bridge CPUs.

Audio streaming is also supported from an iOS device or iTunes to an AirPort Express base station or a 3rd party AirPlay-enabled audio device. Initially this was called **AirTunes**, but it was later renamed to AirPlay when Apple added video support for the Apple TV.

This document describes these protocols, as implemented in Apple TV software version 5.0, iOS 5.1 and iTunes 10.6. They are based on well-known standard networking protocols such as **Multicast DNS**, **HTTP**, **RTSP**, **RTP** or **NTP**, with custom extensions.

All these information have been gathered by using various techniques of reverse engineering, so they might be somewhat inaccurate and incomplete. Moreover, this document does not explain how to circumvent any kind of security implemented by Apple:

- It does not give any RSA keys.
- It does not explain how to decode iTunes videos protected with the **FairPlay** DRM.
- It does not explain the FairPlay authentication (SAPv2.5) used by iOS devices and OS X Mountain Lion to protect audio and screen content.

Please don't e-mail me about this, I won't reply. In fact, none of this is actually required to be able to view media content on Apple TV.

2. Service Discovery

AirPlay does not require any configuration to be able to find compatible devices on the network, thanks to DNS-based service discovery, based on multicast DNS, aka **Bonjour**.

An AirPlay device such as the Apple TV publishes two services. The first one is **RAOP** (Remote Audio Output Protocol), used for audio streaming, and the other one is the AirPlay service, for photo and video content.

2.1. AirTunes service

RAOP service from Apple TV

```
name: 5855CA1AE288@Apple TV
type: _raop._tcp
port: 49152
txt:
 txtvers=1
 ch=2
 cn=0,1,2,3
 da=true
```

```
et=0,3,5
md=0,1,2
pw=false
sv=false
sr=44100
ss=16
tp=UDP
vn=65537
vs=130.14
am=AppleTV2,1
sf=0x4
```

The name is formed using the MAC address of the device and the name of the remote speaker which will be shown by the clients.

The following fields appear in the TXT record:

name	value	description
txtvers	1	TXT record version 1
ch	2	audio channels: stereo
cn	0,1,2,3	audio codecs
et	0,3,5	supported encryption types
md	0,1,2	supported metadata types
pw	false	does the speaker require a password?
sr	44100	audio sample rate: 44100 Hz
ss	16	audio sample size: 16-bit
tp	UDP	supported transport: TCP or UDP
vs	130.14	server version 130.14
am	AppleTV2,1	device model

Audio codecs

cn	description
0	PCM
1	Apple Lossless (ALAC)
2	AAC

3	AAC ELD (Enhanced Low Delay)

Encryption Types

et	description
0	no encryption
1	RSA (AirPort Express)
3	FairPlay
4	MFiSAP (3rd-party devices)
5	FairPlay SAPv2.5

Metadata Types

md	description
0	text
1	artwork
2	progress

2.2. AirPlay Service

AirPlay service

```
name: Apple TV
type: _airplay._tcp
port: 7000
txt:
 deviceid=58:55:CA:1A:E2:88
 features=0x39f7
 model=AppleTV2,1
 srcvers=130.14
```

The following fields are available in the TXT record:

name	value	description
`model`	AppleTV2,1	device model
`deviceid`	58:55:CA:1A:E2:88	MAC address of the device
`features`	0x39f7	bitfield of supported features
`pw`	1	server is password protected

The `pw` field appears only if the AirPlay server is password protected. Otherwise it is not included in the TXT record.
The `features` bitfield allows the following features to be defined:

bit	name	description
0	`Video`	video supported
1	`Photo`	photo supported
2	`VideoFairPlay`	video protected with FairPlay DRM
3	`VideoVolumeControl`	volume control supported for videos
4	`VideoHTTPLiveStreams`	http live streaming supported
5	`Slideshow`	slideshow supported
7	`Screen`	mirroring supported
8	`ScreenRotate`	screen rotation supported
9	`Audio`	audio supported
11	`AudioRedundant`	audio packet redundancy supported
12	`FPSAPv2pt5_AES_GCM`	FairPlay secure auth supported
13	`PhotoCaching`	photo preloading supported

Note that the Apple TV does not support `VideoVolumeControl`. It has probably been introduced for the upcoming Apple television.
The AirPlay server is a **HTTP** server (RFC 2616). Two connections are made to this server, the second one being used as a reverse HTTP connection. This allows a client to receive asynchronous events, such as playback status changes, from a server.
All HTTP requests share some common headers:

name	value	descriptio
`X-Apple-Session-ID`	1bd6ceeb...	UUID for the se
`X-Apple-Device-ID`	0xdc2b61a0ce79	MAC address

The reverse connection looks like this:

client → server

```
POST /reverse
```

```
Upgrade: PTTH/1.0
Connection: Upgrade
X-Apple-Purpose: event
Content-Length: 0
User-Agent: MediaControl/1.0
X-Apple-Session-ID:
1bd6ceeb-fffd-456c-a09c-996053a7a08c
```

server → client

```
HTTP/1.1 101 Switching Protocols
Date: Thu, 23 Feb 2012 17:33:41 GMT
Upgrade: PTTH/1.0
Connection: Upgrade
```

The `X-Apple-Purpose` header makes it clear that this connection is used for sending events to the client, whereas `X-Apple-Session-ID` is used to link this connection to the other (non-reverse) one. Events are delivered using a `POST` request for sending an XML property list to the `/event` location.

3. Photos

Photos are JPEG data transmitted using a `PUT` request to the AirPlay server. They can be displayed immediately, or cached for future use.

3.1. HTTP requests

GET /slideshow-features

A client can fetch the list of available transitions for slideshows. Then it can let the user pick one, before starting a slideshow. The `Accept-Language` header is used to specify in which language the transition names should be.

client → server

```
GET /slideshow-features HTTP/1.1
Accept-Language: English
Content-Length: 0
User-Agent: MediaControl/1.0
X-Apple-Session-ID:
cdda804c-33ae-4a0b-a5f2-f0e532fd5abd
```

server → client

```
HTTP/1.1 200 OK
Date: Thu, 23 Feb 2012 17:33:41 GMT
Content-Type: text/x-apple-plist+xml
Content-Length: 6411

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
 <dict>
  <key>themes</key>
  <array>
   <dict>
    <key>key</key>
    <string>Reflections</string>
    <key>name</key>
    <string>Reflections</string>
   </dict>
   ...
  </array>
 </dict>
</plist>
```

PUT /photo

Send a JPEG picture to the server. The following headers are supported:

name	description
`X-Apple-AssetKey`	UUID for the picture
`X-Apple-Transition`	transition that should be used to show the pictur
`X-Apple-AssetAction`	specify a caching operation

Example 1: show a picture without any transition (for the first time)

client → server

```
PUT /photo HTTP/1.1
X-Apple-AssetKey:
F92F9B91-954E-4D63-BB9A-EEC771ADE6E8
Content-Length: 462848
User-Agent: MediaControl/1.0
X-Apple-Session-ID:
1bd6ceeb-fffd-456c-a09c-996053a7a08c

<JPEG DATA>
```

server → client

```
HTTP/1.1 200 OK
Date: Thu, 23 Feb 2012 17:33:42 GMT
Content-Length: 0
```

Example 2: show a picture using the dissolve transition

client → server

```
PUT /photo HTTP/1.1
X-Apple-AssetKey:
F92F9B91-954E-4D63-BB9A-EEC771ADE6E8
X-Apple-Transition: Dissolve
Content-Length: 462848
User-Agent: MediaControl/1.0
X-Apple-Session-ID:
1bd6ceeb-fffd-456c-a09c-996053a7a08c

<JPEG DATA>
```

server → client

```
HTTP/1.1 200 OK
Date: Thu, 23 Feb 2012 17:33:42 GMT
Content-Length: 0
```

PUT /slideshows/1

Start or stop a slideshow session. When starting, slideshow settings such as the slide duration and selected transition theme are transmitted. The following parameters are sent in an XML property list:

key	type	description
settings.slideDuration	integer	slide duration in seconds
settings.theme	string	selected transition theme
state	string	`playing` or `stopped`

Example: send slideshow settings

client → server

```
PUT /slideshows/1 HTTP/1.1
Content-Type: text/x-apple-plist+xml
Content-Length: 366
User-Agent: MediaControl/1.0
X-Apple-Session-ID:
98a7b246-8e00-49a6-8765-db57165f5b67

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
 <dict>
  <key>settings</key>
  <dict>
   <key>slideDuration</key>
   <integer>3</integer>
   <key>theme</key>
   <string>Classic</string>
  </dict>
  <key>state</key>
  <string>playing</string>
 </dict>
</plist>
```

server → client

```
HTTP/1.1 200 OK
Date: Thu, 08 Mar 2012 16:30:01 GMT
Content-Type: text/x-apple-plist+xml
Content-Length: 181

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
 <dict/>
</plist>
```

POST /stop

Stop a photo or slideshow session.

client → server

```
POST /stop HTTP/1.1
Content-Length: 0
User-Agent: MediaControl/1.0
X-Apple-Session-ID:
1bd6ceeb-fffd-456c-a09c-996053a7a08c
```

server → client

```
HTTP/1.1 200 OK
Date: Thu, 23 Feb 2012 17:33:55 GMT
Content-Length: 0
```

3.2. Events

Photo

This event notifies a client that a photo session has ended. Then the server can safely disconnect.

key	type	description
category	string	**photo**
sessionID	integer	session ID
state	string	**stopped**

Example: stop photo session

server → client

```
POST /event HTTP/1.1
Content-Type: text/x-apple-plist+xml
Content-Length: 277
X-Apple-Session-ID:
1bd6ceeb-fffd-456c-a09c-996053a7a08c

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
 <dict>
  <key>category</key>
  <string>photo</string>
  <key>sessionID</key>
  <integer>38</integer>
  <key>state</key>
  <string>stopped</string>
 </dict>
</plist>
```

client → server

```
HTTP/1.1 200 OK
Content-Length: 0
```

Slideshow

Slideshow events are used to notify the server about the playback state.

key	type	description
category	string	**slideshow**
lastAssetID	integer	last asset ID
sessionID	integer	session ID
state	string	**loading**, **playing** or **stopped**

Example: slideshow is currently playing

server → client

```
POST /event HTTP/1.1
Content-Type: text/x-apple-plist+xml
Content-Length: 371
X-Apple-Session-ID:
f1634b51-5cae-4384-ade5-54f4159a15f1

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
 <dict>
  <key>category</key>
  <string>slideshow</string>
  <key>lastAssetID</key>
  <integer>5</integer>
  <key>sessionID</key>
  <integer>4</integer>
  <key>state</key>
  <string>playing</string>
 </dict>
</plist>
```

client → server

```
HTTP/1.1 200 OK
Content-Length: 0
```

3.3. Photo Caching

AirPlay supports preloading picture data to improve transition latency. This works by preloading a few pictures (most likely the ones before and after the current picture) just after displaying one.
Preloading is achieved using the `cacheOnly` asset action. Upon receiving this request, a server stores the picture in its cache. Later, a client can request the display of this picture using the `displayCached` asset action and the same asset key. This is much faster than a full picture upload because no additional data is transmitted.
When asked for a picture which is no longer in the cache, a server replies with an HTTP 412 error code (Precondition Failed).
Example 1: cache a picture for future display

client → server

```
PUT /photo HTTP/1.1
X-Apple-AssetAction: cacheOnly
X-Apple-AssetKey:
B0DDE2C0-6FDD-48F8-9E5B-29CE0618DF5B
Content-Length: 462848
User-Agent: MediaControl/1.0
X-Apple-Session-ID:
1bd6ceeb-fffd-456c-a09c-996053a7a08c

<JPEG DATA>
```

server → client

```
HTTP/1.1 200 OK
Date: Thu, 23 Feb 2012 17:33:45 GMT
```

```
Content-Length: 0
```

Example 2: show a cached picture

client → server

```
PUT /photo HTTP/1.1
X-Apple-AssetAction: displayCached
X-Apple-AssetKey:
B0DDE2C0-6FDD-48F8-9E5B-29CE0618DF5B
X-Apple-Transition: Dissolve
Content-Length: 0
User-Agent: MediaControl/1.0
X-Apple-Session-ID:
1bd6ceeb-fffd-456c-a09c-996053a7a08c
```

server → client

```
HTTP/1.1 200 OK
Date: Thu, 23 Feb 2012 17:33:45 GMT
Content-Length: 0
```

3.4. Slideshows

Slideshows are using the reverse HTTP connection for asynchronous loading of pictures. Three connections are performed in parallel. The `X-Apple-Purpose` header is set to `slideshow`. A `GET` request to the `/slideshows/1/assets/1` location is issued to fetch a new picture from the AirPlay client. A binary property list with the following parameters is expected as reply:

key	type	description
data	data	JPEG picture
info.id	integer	asset ID
info.key	integer	1

Example: fetch a new picture

server → client

```
GET /slideshows/1/assets/1 HTTP/1.1
```

```
Content-Length: 0
Accept: application/x-apple-binary-plist
X-Apple-Session-ID:
98a7b246-8e00-49a6-8765-db57165f5b67
```

client → server

```
HTTP/1.1 200 OK
Content-Type: application/x-apple-binary-plist
Content-Length: 58932

<BINARY PLIST DATA>

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
 <dict>
  <key>data</key>
  <data>
  ...
  </data>
  <key>info</key>
  <dict>
   <key>id</key>
   <integer>1</integer>
   <key>key</key>
   <string>1</string>
  </dict>
 </dict>
</plist>
```

4. Video

In order to play a video on an AirPlay server, HTTP requests are used to send a video URL, perform scrubbing, change the playback rate and update the timeline.

4.1. HTTP requests

GET /server-info

Fetch general informations about the AirPlay server. These informations are returned as an XML property list, with the following properties:

key	type	value	descripti
deviceid	string	58:55:CA:1A:E2:88	MAC addres
features	integer	14839	0x39f7
model	string	AppleTV2,1	device mode
protovers	string	1.0	protocol vers
srcvers	string	120.2	server versio

The `model`, `deviceid`, `srcvers` and `features` properties are the same as broadcasted by the mDNS AirPlay service.

Example: fetch server informations

client → server

```
GET /server-info HTTP/1.1
X-Apple-Device-ID: 0xdc2b61a0ce79
Content-Length: 0
User-Agent: MediaControl/1.0
X-Apple-Session-ID:
1bd6ceeb-fffd-456c-a09c-996053a7a08c
```

server → client

```
HTTP/1.1 200 OK
Date: Thu, 23 Feb 2012 17:33:41 GMT
Content-Type: text/x-apple-plist+xml
Content-Length: 427

<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
 <dict>
  <key>deviceid</key>
  <string>58:55:CA:1A:E2:88</string>
  <key>features</key>
  <integer>14839</integer>
  <key>model</key>
  <string>AppleTV2,1</string>
  <key>protovers</key>
  <string>1.0</string>
  <key>srcvers</key>
  <string>120.2</string>
 </dict>
</plist>
```

POST /play

Start video playback. The body contains the following parameters:

name	type	description
`Content-Location`	URL	URL for the video
`Start-Position`	float	starting position between 0 and 1

MP4 movies are supported using progressive download. HTTP Live Streaming might be supported as well, as indicated by the `VideoHTTPLiveStreams` feature flag. The relative starting position, a float value between 0 (beginning) and 1 (end) is used to start playing a video at the exact same position as it was on the client.
A binary property list can also be used instead of text parameters, with content type `application/x-apple-binary-plist`.

Example 1: video playback from iTunes

client → server

```
POST /play HTTP/1.1
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Content-Length: 163
Content-Type: text/parameters

Content-Location:
http://192.168.1.18:3689/airplay.mp4?database-spec='d
map.persistentid:0x63b5e5c0c201542e'&item-spec='dmap.
itemid:0x21d'
Start-Position: 0.174051
```

server → client

```
HTTP/1.1 200 OK
Date: Mon, 08 Mar 2012 18:08:25 GMT
Content-Length: 0
```

Example 2: video playback from iPhone

client → server

```
POST /play HTTP/1.1
X-Transmit-Date: 2012-03-16T14:20:39.656533Z
Content-Type: application/x-apple-binary-plist
Content-Length: 491
User-Agent: MediaControl/1.0
X-Apple-Session-ID:
368e90a4-5de6-4196-9e58-9917bdd4ffd7

<BINARY PLIST DATA>

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
```

```
 <dict>
  <key>Content-Location</key>

<string>http://redirector.c.youtube.com/videoplayback
?...</string>
  <key>Start-Position</key>
  <real>0.024613151326775551</real>
 </dict>
</plist>
```
server → client
```
HTTP/1.1 200 OK
```
POST /scrub

Seek at an arbitrary location in the video. The `position` argument is a float value representing the location in seconds.

Example: seek to about 20 seconds

client → server
```
POST /scrub?position=20.097000 HTTP/1.1
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Content-Length: 0
```
server → client
```
HTTP/1.1 200 OK
Date: Mon, 08 Mar 2012 18:08:42 GMT
Content-Length: 0
```
POST /rate

Change the playback rate. The `value` argument is a float value representing the playback rate: 0 is paused, 1 is playing at the normal speed.

Example: pause playback

client → server

```
POST /rate?value=0.000000 HTTP/1.1
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Content-Length: 0
```

server → client

```
HTTP/1.1 200 OK
Date: Mon, 08 Mar 2012 18:08:37 GMT
Content-Length: 0
```

POST /stop

Stop playback.

Example: stop playback

client → server

```
POST /stop HTTP/1.1
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Content-Length: 0
```

server → client

```
HTTP/1.1 200 OK
Date: Mon, 08 Mar 2012 18:09:06 GMT
Content-Length: 0
```

GET /scrub

Retrieve the current playback position. This can be called repeatedly to update a timeline on the client. The following parameters are returned:

name	type	description
`duration`	float	duration in seconds
`position`	float	position in seconds

Example: fetch current playback progress

client → server

```
GET /scrub HTTP/1.1
```

```
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Content-Length: 0
```

server → client

```
HTTP/1.1 200 OK
Date: Mon, 08 Mar 2012 18:08:31 GMT
Content-Type: text/parameters
Content-Length: 40

duration: 83.124794
position: 14.467000
```

GET /playback-info

Retrieve playback informations such as position, duration, rate, buffering status and more. An XML property list is returned with the following parameters:

key	type	description
`duration`	real	playback duration in secor
`position`	real	playback position in secon
`rate`	real	playback rate
`readyToPlay`	boolean	ready to play
`playbackBufferEmpty`	boolean	buffer empty
`playbackBufferFull`	boolean	buffer full
`playbackLikelyToKeepUp`	boolean	playback likely to keep up
`loadedTimeRanges`	array	array of loaded time range
`seekableTimeRanges`	array	array of seekable time ran

Ranges are defined as dictionaries with the following keys:

key	type	description
`start`	real	range start time in seconds
`duration`	real	range duration in seconds

Example: get playback info

client → server

```
GET /playback-info HTTP/1.1
Content-Length: 0
User-Agent: MediaControl/1.0
X-Apple-Session-ID:
24b3fd94-1b6d-42b1-89a3-47108bfbac89
```

server → client

```
HTTP/1.1 200 OK
Date: Fri, 16 Mar 2012 15:31:42 GMT
Content-Type: text/x-apple-plist+xml
Content-Length: 801
X-Transmit-Date: 2012-03-16T15:31:42.607066Z

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
 <dict>
  <key>duration</key> <real>1801</real>
  <key>loadedTimeRanges</key>
  <array>
   <dict>
    <key>duration</key> <real>51.541130402</real>
    <key>start</key> <real>18.118717650000001</real>
   </dict>
  </array>
  <key>playbackBufferEmpty</key> <true/>
  <key>playbackBufferFull</key> <false/>
  <key>playbackLikelyToKeepUp</key> <true/>
  <key>position</key> <real>18.043869775000001</real>
  <key>rate</key> <real>1</real>
  <key>readyToPlay</key> <true/>
  <key>seekableTimeRanges</key>
  <array>
   <dict>
    <key>duration</key>
    <real>1801</real>
    <key>start</key>
    <real>0.0</real>
   </dict>
  </array>
 </dict>
</plist>
```

PUT /setProperty

Set playback property. The property name is sent as query argument. The following properties are defined:

argument	description
`forwardEndTime`	forward end time
`reverseEndTime`	reverse end time

Example: set forward end time

client → server

```
PUT /setProperty?forwardEndTime HTTP/1.1
Content-Type: application/x-apple-binary-plist
Content-Length: 96
User-Agent: MediaControl/1.0
X-Apple-Session-ID:
24b3fd94-1b6d-42b1-89a3-47108bfbac89

<BINARY PLIST DATA>

<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
 <dict>
```

```
  <key>value</key>
  <dict>
   <key>epoch</key> <integer>0</integer>
   <key>flags</key> <integer>0</integer>
   <key>timescale</key> <integer>0</integer>
   <key>value</key> <integer>0</integer>
  </dict>
 </dict>
</plist>
```

server → client

```
HTTP/1.1 200 OK
Date: Fri, 16 Mar 2012 15:23:11 GMT
Content-Type: application/x-apple-binary-plist
Content-Length: 58

<BINARY PLIST DATA>

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
 <dict>
  <key>errorCode</key>
  <integer>0</integer>
 </dict>
</plist>
```

GET /getProperty

Get playback property. The property name is sent as query argument. The following properties are defined:

argument	description
`playbackAccessLog`	playback access log

`playbackErrorLog`	playback error log

Example: get playback access log

client → server

```
POST /getProperty?playbackAccessLog HTTP/1.1
Content-Type: application/x-apple-binary-plist
Content-Length: 0
User-Agent: MediaControl/1.0
X-Apple-Session-ID:
24b3fd94-1b6d-42b1-89a3-47108bfbac89
```

server → client

```
HTTP/1.1 200 OK
Date: Fri, 16 Mar 2012 15:31:42 GMT
Content-Type: application/x-apple-binary-plist
Content-Length: 530

<BINARY PLIST DATA>

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
 <dict>
  <key>errorCode</key>
  <integer>0</integer>
  <key>value</key>
  <array>
   <dict>
    <key>bytes</key> <integer>1818336</integer>
    <key>c-duration-downloaded</key> <real>70</real>
    <key>c-duration-watched</key>
<real>18.154102027416229</real>
    <key>c-frames-dropped</key> <integer>0</integer>
    <key>c-observed-bitrate</key>
<real>14598047.302367469</real>
    <key>c-overdue</key> <integer>0</integer>
    <key>c-stalls</key> <integer>0</integer>
    <key>c-start-time</key> <real>0.0</real>
    <key>c-startup-time</key>
<real>0.27732497453689575</real>
    <key>cs-guid</key>
<string>B475F105-78FD-4200-96BC-148BAB6DAC11</string>
    <key>date</key> <date>2012-03-16T15:31:24Z</date>
    <key>s-ip</key> <string>213.152.6.89</string>
    <key>s-ip-changes</key> <integer>0</integer>
    <key>sc-count</key> <integer>7</integer>
    <key>uri</key>
<string>http://devimages.apple.com/iphone/samples/bip
bop/gear1/prog_index.m3u8</string>
   </dict>
  </array>
 </dict>
</plist>
```

4.2. Events

This event is used to send the playback state to the client:

key	type	description
category	string	`video`
sessionID	integer	session id
state	string	`loading`, `playing`, `paused` or `stopped`

Example: notify the client that video playback is paused

server → client

```
POST /event HTTP/1.1
Content-Type: application/x-apple-plist
```

```
Content-Length: 321
X-Apple-Session-ID:
00000000-0000-0000-0000-000000000000

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
  "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
 <dict>
  <key>category</key>
  <string>video</string>
  <key>sessionID</key>
  <integer>13</integer>
  <key>state</key>
  <string>paused</string>
 </dict>
</plist>
```

client → server

```
HTTP/1.1 200 OK
Content-Length: 0
Date: Mon, 08 Mar 2012 18:07:43 GMT
```

5. Audio

Audio streaming is supported using the **RTSP** protocol (RFC 2326).

5.1. RTSP requests

OPTIONS

The `OPTIONS` request asks the RTSP server for its supported methods. Apple TV supports the following methods: `ANNOUNCE`, `SETUP`, `RECORD`, `PAUSE`, `FLUSH`, `TEARDOWN`, `OPTIONS`, `GET_PARAMETER`, `SET_PARAMETER`, `POST` and `GET`.

client → server

```
OPTIONS * RTSP/1.0
CSeq: 3
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Client-Instance: 56B29BB6CB904862
DACP-ID: 56B29BB6CB904862
Active-Remote: 1986535575
```

server → client

```
RTSP/1.0 200 OK
Public: ANNOUNCE, SETUP, RECORD, PAUSE, FLUSH, TEARDOWN,
OPTIONS,
GET_PARAMETER, SET_PARAMETER, POST, GET
Server: AirTunes/130.14
CSeq: 3
```

ANNOUNCE

The `ANNOUNCE` request tells the RTSP server about stream properties using SDP (RFC 4566). Codec informations and encryption keys are of particular interest.

Example 1: `ANNOUNCE` for **Apple Lossless** audio from iTunes

client → server

```
ANNOUNCE rtsp://fe80::217:f2ff:fe0f:e0f6/3413821438
RTSP/1.0
CSeq: 3
Content-Type: application/sdp
Content-Length: 348
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Client-Instance: 56B29BB6CB904862
DACP-ID: 56B29BB6CB904862
```

```
Active-Remote: 1986535575

v=0
o=iTunes 3413821438 0 IN IP4 fe80::217:f2ff:fe0f:e0f6
s=iTunes
c=IN IP4 fe80::5a55:caff:fe1a:e187
t=0 0
m=audio 0 RTP/AVP 96
a=rtpmap:96 AppleLossless
a=fmtp:96 352 0 16 40 10 14 2 255 0 0 44100
a=fpaeskey:RlBMWQECAQAAAAA8AAAAAPFOnNe+zWb5/n4L5KZkE2
AAAAAQlDx69reTdwHF9LaNmhiRURTAbcL4brYAceAkZ49YirXm62N
4
a=aesiv:5b+YZi9Ikb845BmNhaVo+Q
```

server → client

```
RTSP/1.0 200 OK
Server: AirTunes/130.14
CSeq: 3
```

Example 2: ANNOUNCE for AAC audio from an iOS device

client → server

```
ANNOUNCE rtsp://192.168.1.45/2699324803567405959
RTSP/1.0
X-Apple-Device-ID: 0xa4d1d2800b68
CSeq: 16
DACP-ID: 14413BE4996FEA4D
Active-Remote: 2543110914
Content-Type: application/sdp
Content-Length: 331

v=0
o=AirTunes 2699324803567405959 0 IN IP4 192.168.1.5
s=AirTunes
c=IN IP4 192.168.1.5
```

```
t=0 0
m=audio 0 RTP/AVP 96
a=rtpmap:96 mpeg4-generic/44100/2
a=fmtp:96
a=fpaeskey:RlBMWQECAQAAAAA8AAAAAOG6c4aMdLkXAX+lbjp7Eh
gAAAAQeX5uqGyYkBmJX+gd5ANEr+amI8urqFmvcNo87pR0BXGJ4eL
f
a=aesiv:VZTaHn4wSJ84Jjzlb94m0Q==
a=min-latency:11025
```

server → client

```
RTSP/1.0 200 OK
Server: AirTunes/130.14
CSeq: 16
```

Example 3: ANNOUNCE for AAC-ELD audio and H.264 video from an iOS device

client → server

```
ANNOUNCE rtsp://192.168.1.45/846700446248110360
RTSP/1.0
X-Apple-Device-ID: 0xa4d1d2800b68
CSeq: 27
DACP-ID: 14413BE4996FEA4D
Active-Remote: 2543110914
Content-Type: application/sdp
Content-Length: 415

v=0
o=AirTunes 846700446248110360 0 IN IP4 192.168.1.5
s=AirTunes
c=IN IP4 192.168.1.5
t=0 0
m=audio 0 RTP/AVP 96
a=rtpmap:96 mpeg4-generic/44100/2
a=fmtp:96 mode=AAC-eld; constantDuration=480
```

```
a=fpaeskey:RlBMWQECAQAAAAA8AAAAAKKp+t27A+686xfviEphhw
8AAAAQE/3LSqv9MHgnEKxkbKh1buE9+ylKg0YuqcyAC7fT0EqJNtd
q
a=aesiv:i/a3nUKYNDSIPP2fC+UKGQ==
a=min-latency:4410
m=video 0 RTP/AVP 97
a=rtpmap:97 H264
a=fmtp:97
```

server → client

```
RTSP/1.0 200 OK
Server: AirTunes/130.14
CSeq: 27
```

SETUP

The `SETUP` request initializes a record session. It sends all the necessary transport informations. Three UDP channels are setup:

channel	description
server	audio data
control	sync and retransmit requests
timing	master clock sync

Example: setup a record session

client → server

```
SETUP rtsp://fe80::217:f2ff:fe0f:e0f6/3413821438
RTSP/1.0
CSeq: 4
Transport:
RTP/AVP/UDP;unicast;interleaved=0-1;mode=record;contr
ol_port=6001;timing_port=6002
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Client-Instance: 56B29BB6CB904862
DACP-ID: 56B29BB6CB904862
```

```
Active-Remote: 1986535575
```

server → client

```
RTSP/1.0 200 OK
Transport:
RTP/AVP/UDP;unicast;mode=record;server_port=53561;con
trol_port=63379;timing_port=50607
Session: 1
Audio-Jack-Status: connected
Server: AirTunes/130.14
CSeq: 4
```

RECORD

The `RECORD` request starts the audio streaming. The `RTP-Info` header contains the following parameters:

name	size	description
`seq`	16-bit	initial RTP sequence number
`rtptime`	32-bit	initial RTP timestamp

Example: start audio stream

client → server

```
RECORD rtsp://fe80::217:f2ff:fe0f:e0f6/3413821438
RTSP/1.0
CSeq: 5
Session: 1
Range: npt=0-
RTP-Info: seq=20857;rtptime=1146549156
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Client-Instance: 56B29BB6CB904862
DACP-ID: 56B29BB6CB904862
Active-Remote: 1986535575
```

server → client

```
RTSP/1.0 200 OK
Audio-Latency: 2205
```

```
Server: AirTunes/130.14
CSeq: 5
```

FLUSH

The `FLUSH` request stops the streaming.
Example: pause the audio stream

client → server

```
FLUSH rtsp://fe80::217:f2ff:fe0f:e0f6/3413821438
RTSP/1.0
CSeq: 31
Session: 1
RTP-Info: seq=25009;rtptime=1148010660
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Client-Instance: 56B29BB6CB904862
DACP-ID: 56B29BB6CB904862
Active-Remote: 1986535575
```

server → client

```
RTSP/1.0 200 OK
RTP-Info: rtptime=1147914212
Server: AirTunes/130.14
CSeq: 31
```

TEARDOWN

The `TEARDOWN` request ends the RTSP session.
Example: close session 1

client → server

```
TEARDOWN rtsp://fe80::217:f2ff:fe0f:e0f6/3413821438
RTSP/1.0
CSeq: 32
Session: 1
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
```

```
Client-Instance: 56B29BB6CB904862
DACP-ID: 56B29BB6CB904862
Active-Remote: 1986535575
```

server → client

```
RTSP/1.0 200 OK
Server: AirTunes/130.14
CSeq: 32
```

5.2. RTP Streams

Audio packets are fully RTP compliant. Control and timing packets, however, do not seem to be fully compliant with the RTP standard.
The following payload types are defined:

payload type	port	description
82	`timing_port`	timing request
83	`timing_port`	timing reply
84	`control_port`	time sync
85	`control_port`	retransmit request
86	`control_port`	retransmit reply
96	`server_port`	audio data

Audio packets

Audio data is sent using the `DynamicRTP-Type-96` payload type. The `Marker` bit is set on the first packet sent after `RECORD` or `FLUSH` requests. The RTP payload contains optionally encrypted audio data.
Example: encrypted audio packet

client → server

```
0000   80 e0 b1 91 f7 79 16 c2 e8 bb 6b 2c bb 5c 8e 51
0010   aa 7c d2 96 00 c3 fd 60 eb ae 6e 41 31 38 fe ae
....
03e0   cb 1c 73 bf e7 05 93 30 fa 85 7f 32 77 8d a8 97
03f0   a0 c7 c8 78 7b e5 81 a1 4f b4 3e a3 43 db 7c
```

```
Real-Time Transport Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 .... = Extension: False
    .... 0000 = Contributing source identifiers count:
0
    1... .... = Marker: True
    Payload type: DynamicRTP-Type-96 (96)
    Sequence number: 45457
    Timestamp: 4151908034
    Synchronization Source identifier: 0xe8bb6b2c
(3904596780)
    Payload:
bb5c8e51aa7cd29600c3fd60ebae6e413138feae909b44f1...
```

Sync packets

Sync packets are sent once per second to the control port. They are used to correlate the RTP timestamps currently used in the audio stream to the NTP time used for clock synchronization. Payload type is 84, the `Marker` bit is always set and the `Extension` bit is set on the first packet after `RECORD` or `FLUSH` requests. The `SSRC` field is not included in the RTP header.

bytes	description
8	RTP header without `SSRC`
8	current NTP time
4	RTP timestamp for the next audio packet

Example: sync packet

client → server

```
0000   80 d4 00 04 c7 cd 11 a8 83 ab 1c 49 2f e4 22 e2
0010   c7 ce 3f 1f
```

```
Real-Time Transport Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 .... = Extension: False
    .... 0000 = Contributing source identifiers count:
0
    1... .... = Marker: True
    Payload type: Unassigned (84)
    Sequence number: 4
    Timestamp: 3352105384
    Synchronization Source identifier: 0x83ab1c49
(2209029193)
    Payload: 2fe422e2c7ce3f1f
```

Retransmit packets

AirTunes supports resending audio packets which have been lost. Payload type is 85 for retransmit queries, the `Marker` bit is always set and the `SSRC` field is not included in the RTP header.

bytes	description
8	RTP header without `SSRC`
2	sequence number for the first lost packet
2	number of lost packets

Retransmit replies have payload type 86, with a full audio RTP packet after the sequence number.

Timing packets

Timing packets are used to synchronize a master clock for audio. This is useful for clock recovery and precise synchronization of several devices playing the same audio stream.
Timing packets are sent at 3 second intervals. They always have the `Marker` bit set, and payload type 82 for

queries and 83 for replies. The **SSRC** field is not included in the RTP header, so it takes only 8 bytes, followed by three **NTP** timestamps:

bytes	description
8	RTP header without **SSRC**
8	origin timestamp
8	receive timestamp
8	transmit timestamp

Example: timing query/reply

server → client

```
0000   80 d2 00 07 00 00 00 00 00 00 00 00 00 00 00 00
0010   00 00 00 00 00 00 00 00 83 c1 17 cc af ba 9b 32

Real-Time Transport Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 .... = Extension: False
    .... 0000 = Contributing source identifiers count:
0
    1... .... = Marker: True
    Payload type: Unassigned (82)
    Sequence number: 7
    Timestamp: 0
    Synchronization Source identifier: 0x00000000 (0)
    Payload: 000000000000000000000000083c117ccafba9b32
```

client → server

```
0000   80 d3 00 07 00 00 00 00 83 c1 17 cc af ba 9b 32
0010   83 c1 17 cc b0 12 ce b6 83 c1 17 cc b0 14 10 47

Real-Time Transport Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 .... = Extension: False
    .... 0000 = Contributing source identifiers count:
0
    1... .... = Marker: True
    Payload type: Unassigned (83)
    Sequence number: 7
    Timestamp: 0
    Synchronization Source identifier: 0x83c117cc
(2210469836)
    Payload: afba9b3283c117ccb012ceb683c117ccb0141047
```

5.3. Volume Control

Audio volume can be changed using a `SET_PARAMETER` request. The volume is a float value representing the audio attenuation in dB. A value of –144 means the audio is muted. Then it goes from –30 to 0.

Example: set audio volume

client → server

```
SET_PARAMETER
rtsp://fe80::217:f2ff:fe0f:e0f6/3413821438 RTSP/1.0
CSeq: 6
Session: 1
Content-Type: text/parameters
Content-Length: 20
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Client-Instance: 56B29BB6CB904862
DACP-ID: 56B29BB6CB904862
Active-Remote: 1986535575

volume: -11.123877
```

server → client

```
RTSP/1.0 200 OK
Server: AirTunes/130.14
```

```
CSeq: 6
```

5.4. Metadata

Metadata for the current track are sent using `SET_PARAMETER` requests. This allows the Apple TV to show the track name, artist, album, cover artwork and timeline. The `RTP-Info` header contains a `rtptime` parameter with the RTP timestamp corresponding to the time from which the metadata is valid.

Track Informations

Informations about the current track are sent in the **DAAP** (Digital Audio Access Protocol) format, with `application/x-dmap-tagged` content type.
The following DAAP attributes are displayed on Apple TV:

attribute	description
`dmap.itemname`	track name
`daap.songartist`	artist
`daap.songalbum`	album

Example: send track informations

client → server

```
SET_PARAMETER
rtsp://fe80::217:f2ff:fe0f:e0f6/3413821438 RTSP/1.0
CSeq: 8
Session: 1
Content-Type: application/x-dmap-tagged
Content-Length: 3242
RTP-Info: rtptime=1146549156
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Client-Instance: 56B29BB6CB904862
DACP-ID: 56B29BB6CB904862
```

```
Active-Remote: 1986535575

<DMAP DATA>
```

server → client

```
RTSP/1.0 200 OK
Server: AirTunes/130.14
CSeq: 8
```

Cover Artwork

Artworks are sent as JPEG pictures, with `image/jpeg` content type.

Example: send cover artwork

client → server

```
SET_PARAMETER
rtsp://fe80::217:f2ff:fe0f:e0f6/3413821438 RTSP/1.0
CSeq: 9
Session: 1
Content-Type: image/jpeg
Content-Length: 34616
RTP-Info: rtptime=1146549156
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Client-Instance: 56B29BB6CB904862
DACP-ID: 56B29BB6CB904862
Active-Remote: 1986535575

<JPEG DATA>
```

server → client

```
RTSP/1.0 200 OK
Server: AirTunes/130.14
CSeq: 9
```

Playback Progress

Playback progress is sent as **`text/parameters`**, with a **`progress`** parameter representing three absolute RTP timestamps values: **`start/curr/end`**.

timestamp	description
`start`	beginning of the current track
`curr`	current playback position
`end`	end of the current track

The relative position and track duration can be computed as follows:

- **`position = rtptime_to_sec(curr - start)`**
- **`duration = rtptime_to_sec(end - start)`**

<u>**Example:**</u> send playback progress

client → server

```
SET_PARAMETER
rtsp://fe80::217:f2ff:fe0f:e0f6/3413821438 RTSP/1.0
CSeq: 10
Session: 1
Content-Type: text/parameters
Content-Length: 44
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X
10.7.3)
AppleWebKit/535.18.5
Client-Instance: 56B29BB6CB904862
DACP-ID: 56B29BB6CB904862
Active-Remote: 1986535575

progress: 1146221540/1146549156/1195701740
```

server → client

```
RTSP/1.0 200 OK
Server: AirTunes/130.14
CSeq: 10
```

5.5. AirPort Express Authentication

Sending audio data to the AirPort Express requires a **RSA** based authentication. All binary data are encoded using **Base64** (RFC 4648) without padding.

Client side

- In the `ANNOUNCE` request, the client sends a 128-bit random number in the `Apple-Challenge` header.
- A 128-bit **AES** key is generated, encrypted with the RSA public key using the **OAEP** encryption scheme, and sent along with an initialization vector in the `rsaaeskey` and `aesiv` SDP attributes.

Server side

- The AirPort Express decrypts the AES key with its RSA private key, it will be used to decrypt the audio payload.
- The AirPort Express signs the `Apple-Challenge` number with its RSA private key using the **PKCS#1** signature scheme and send the result in the `Apple-Response` header.

Client side

- The client decrypts the `Apple-Response` value with the RSA public key, and checks that it is the same random number it has previously generated.

Example: AirPort Express challenge/response

client → server

```
ANNOUNCE rtsp://10.0.1.101/3172942895 RTSP/1.0
CSeq: 1
```

```
Content-Type: application/sdp
Content-Length: 567
User-Agent: iTunes/4.6 (Windows; N)
Client-Instance: 9FF35780A8BC8D2B
Apple-Challenge: 09KF45soMYmvj6dpsUGiIg

v=0
o=iTunes 3172942895 0 IN IP4 10.0.1.101
s=iTunes
c=IN IP4 10.0.1.103
t=0 0
m=audio 0 RTP/AVP 96
a=rtpmap:96 AppleLossless
a=fmtp:96 4096 0 16 40 10 14 2 255 0 0 44100
a=rsaaeskey:5QYIqmdZGTONY5SHjEJrqAhaa0W9wzDC5i6q221md
GZJ5ubO6Kg

yhC6U83wpY87TFdPRdfPQl2kVC7+Uefmx1bXdIUoO7ZcJsqMbgtje
4w2JQw0b

Uw2BlzNPmVGQOxfdpGc3LXZzNE0jI1D4conUEiW6rrzikXBhk7Y/i
2naw13ayy

xaSwtkiJ0ltBQGYGErbV2tx43QSNj7OOJIG9GrF2GZZ6/UHo4VH+Z
XgQ4NZvP/

QXPCsLutZsvusFDzIEq7TN1fveINOiwrzlN+bckEixvhXlvoQTWE2
tjbmQYhMvO

FIly5gNbZiXi0l5AdolX4jDC2vndFHqWDks/3sPikNg
a=aesiv:zcZmAZtqh7uGcEwPXk0QeA
```

server → client

```
RTSP/1.0 200 OK
```

```
CSeq: 1
Apple-Response:
u+msU8Cc7KBrVPjI/Ir8fOL8+C5D3Jsw1+acaW3MNTndrTQAeb/a

5m1OUVBX6wb/DYQGY+b28ksSwBjN0nFOk4Y2cODEf83FAh7B

mkLpmpkpplp7zVXQ+Z9DcB6gC60ZsS3t98aoR7tSzVLKZNgi2X2sC
+vGsz

utQxX03HK008VjcdngHv3g1p2knoETd07T6eVfZCmPqp6Ga7Dj8VI
Ij/GEP3

AjjDx31JnQBXUDmxM484YXLXZjWFXCiY8GJt6whjf7/2c3rIoT3Z7
PQpEvPmM

1MXU9cv4NL59Y/q0OAVQ38foOz7eGAhfvjOsCnHU25aik7/7ToIYt
1tyVtap/kA
Audio-Jack-Status: connected; type=analog
```

5.6. Remote Control

Audio speakers can send commands to the AirPlay client to change the current track, pause and resume playback, shuffle the playlist, and more. This uses a subset of **DACP** (Digital Audio Control Protocol). An AirPlay client advertises this capability by including a `DACP-ID` header in its RTSP requests, with a 64-bit ID for the DACP server. An `Active-Remote` header is included as well, serving as an authentication token. The AirPlay server needs to browse the mDNS `_dacp._tcp` services for a matching DACP server. Server names look like `iTunes_Ctrl_$ID`.

DACP service from iTunes

```
name: iTunes_Ctrl_56B29BB6CB904862
```

```
type: _dacp._tcp
port: 3689
txt:
 txtvers=1
 Ver=131075
 DbId=63B5E5C0C201542E
 OSsi=0x1F5
```

Once the DACP server has been identified, HTTP requests can be sent to the corresponding service port. The `Active-Remote` header must be included in these requests, so no additional pairing is required. The location for remote control commands is `/ctrl-int/1/$CMD`. The following commands are available:

command	description
`beginff`	begin fast forward
`beginrew`	begin rewind
`mutetoggle`	toggle mute status
`nextitem`	play next item in playlist
`previtem`	play previous item in playlist
`pause`	pause playback
`playpause`	toggle between play and pause
`play`	start playback
`stop`	stop playback
`playresume`	play after fast forward or rewind
`shuffle_songs`	shuffle playlist
`volumedown`	turn audio volume down
`volumeup`	turn audio volume up

Example: send a pause command

server → client

```
GET /ctrl-int/1/pause HTTP/1.1
Host: starlight.local.
Active-Remote: 1986535575
```

client → server

```
HTTP/1.1 204 No Content
Date: Tue, 06 Mar 2012 16:38:51 GMT
DAAP-Server: iTunes/10.6 (Mac OS X)
Content-Type: application/x-dmap-tagged
Content-Length: 0
```

6. Screen Mirroring

Screen mirroring is achieved by transmitting an H.264 encoded video stream over a TCP connection. This stream is packetized with a 128-byte header. AAC-ELD audio is sent using the AirTunes protocol. As for the master clock, it is synchronized using NTP. Moreover, as soon as a client starts a video playback, a standard AirPlay connection is made to send the video URL, and mirroring is stopped. This avoids decoding and re-encoding the video, which would incur a quality loss.

6.1. HTTP requests

Screen mirroring does not use the standard AirPlay service. Instead it connects to an apparently hard-coded port 7100. This is a HTTP server which supports the following requests:

GET /stream.xml

Retrieve information about the server capabilities. The server sends an XML property list with the following properties:

key	type	value	description
`height`	integer	720	vertical resolution
`width`	integer	1280	horizontal resolution
`overscanned`	boolean	true	is the display overscanned?

refreshRate	real	0.01666...	refresh rate 60 Hz (1/60)
version	string	130.14	server version

These properties tell us that the AirPlay server is connected to a 1280x720, 60 Hz, overscanned display.

Example: fetch mirroring server informations

client → server

```
GET /stream.xml HTTP/1.1
Content-Length: 0
```

server → client

```
HTTP/1.1 200 OK
Date: Mon, 08 Mar 2012 15:30:27 GMT
Content-Type: text/x-apple-plist+xml
Content-Length: 411

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
 <dict>
  <key>height</key>
  <integer>720</integer>
  <key>overscanned</key>
  <true/>
  <key>refreshRate</key>
  <real>0.016666666666666666</real>
  <key>version</key>
  <string>130.14</string>
  <key>width</key>
  <integer>1280</integer>
 </dict>
</plist>
```

POST /stream

Start the live video transmission. The client sends a binary property list with information about the stream, immediately followed by the stream itself. At this point, the connection is no longer a valid HTTP connection.
The following parameters are sent:

key	type	value	description
deviceID	integer	181221086727016	MAC address (A4:D1:D2:80
sessionID	integer	-808788724	session ID (0xcfcadd0c)
version	string	130.16	server version
param1	data	(72 bytes)	AES key, encrypted with Fai
param2	data	(16 bytes)	AES initialization vector
latencyMs	integer	90	video latency in ms
fpsInfo	array		
timestampInfo	array		

The `param1` and `param2` parameters are optional.
As soon as the server receives a `/stream` request, it will send NTP requests to the client on port 7010, which seems hard-coded as well. The client needs to export its master clock there, which will be used for audio/video synchronization and clock recovery.
Example: send stream information

client → server

```
POST /stream HTTP/1.1
X-Apple-Device-ID: 0xa4d1d2800b68
Content-Length: 503

<BINARY PLIST DATA>

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
```

```
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
 <dict>
  <key>deviceID</key>
  <integer>181221086727016</integer>
  <key>fpsInfo</key>
  <array>
   <dict> <key>name</key> <string>SubS</string> </dict>
   <dict> <key>name</key> <string>B4En</string> </dict>
   <dict> <key>name</key> <string>EnDp</string> </dict>
   <dict> <key>name</key> <string>IdEn</string> </dict>
   <dict> <key>name</key> <string>IdDp</string> </dict>
   <dict> <key>name</key> <string>EQDp</string> </dict>
   <dict> <key>name</key> <string>QueF</string> </dict>
   <dict> <key>name</key> <string>Sent</string> </dict>
  </array>
  <key>latencyMs</key>
  <integer>90</integer>
  <key>param1</key>
  <data>

RlBMWQECAQAAAAA8AAAAANvKuDizduszL1hG9IvIk+AAAAAQukdPJ
5Jw/gGBAl22WZdF
  m9ujZEGIV7jm3ZByWm51HjpDwjYY
  </data>
  <key>param2</key>
  <data>
  3qpOHtYWbBPyEWPnGt1BuQ==
  </data>
  <key>sessionID</key>
  <integer>-808788724</integer>
  <key>timestampInfo</key>
  <array>
```

```
   <dict> <key>name</key> <string>SubSu</string>
</dict>
   <dict> <key>name</key> <string>BePxT</string>
</dict>
   <dict> <key>name</key> <string>AfPxT</string>
</dict>
   <dict> <key>name</key> <string>BefEn</string>
</dict>
   <dict> <key>name</key> <string>EmEnc</string>
</dict>
   <dict> <key>name</key> <string>QueFr</string>
</dict>
   <dict> <key>name</key> <string>SndFr</string>
</dict>
  </array>
  <key>version</key>
  <string>130.16</string>
 </dict>
</plist>
```

6.2. Stream Packets

The video stream is packetized using 128-byte headers, followed by an optional payload. Only the first 64 bytes of headers seem to be used. Headers start with the following little-endian fields:

size	description
4 bytes	payload size
2 bytes	payload type
2 bytes	0x1e if type = 2, else 6
8 bytes	NTP timestamp

There are 3 types of packets:

type	description
0	video bitstream

1	codec data
2	heartbeat

Codec Data

This packet contains the H.264 extra data in **avcC** format (ISO/IEC 14496:15). It is sent at the beginning of the stream, each time the video properties might change, when screen orientation changes, and when the screen is turned on or off.

H.264 codec data from iPad

```
0000   01 64 c0 28 ff e1 00 10 67 64 c0 28 ac 56 20 0d
0010   81 4f e5 9b 81 01 01 01 01 00 04 28 ee 3c b0
```

The H.264 codec data is interpreted as follows:

size	value	description
1 byte	1	version
1 byte	100	profile (high)
1 byte	0xc0	compatibility
1 byte	40	level (4.0)
6 bits	0x3f	reserved
2 bits	3	NAL units length size – 1
3 bits	0x7	reserved
5 bits	1	number of SPS
2 bytes	16	length of SPS
16 bytes	…	Sequence parameter set
1 byte	1	number of PPS
2 bytes	4	length of PPS
4 bytes	…	Picture parameter set

Codec data packet from iPad

```
0000   1f 00 00 00 01 00 06 00 1d 9a 9f 59 ef de 00 00
0010   00 00 58 44 00 00 22 44 00 00 00 00 00 00 00 00
0020   00 00 00 00 00 00 00 00 00 00 58 44 00 00 22 44
0030   00 00 50 43 00 00 10 42 00 c0 57 44 00 c0 21 44
0040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
0050    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080    01 64 c0 28 ff e1 00 10 67 64 c0 28 ac 56 20 0d
0090    81 4f e5 9b 81 01 01 01 01 00 04 28 ee 3c b0
```

Video Bitstream

This packet contains the video bitstream to be decoded. The payload can be optionally AES encrypted. The NTP timestamp found in the header serves as presentation timestamp.

Video bitstream packet from iPad

```
0000    c8 08 00 00 00 00 06 00 e9 e6 f5 ac 60 e0 00 00
0010    58 37 6e f9 40 01 00 00 00 00 00 00 00 00 00 00
0020    00 00 00 00 00 00 00 00 00 00 58 44 00 00 22 44
0030    00 00 50 43 00 00 10 42 00 c0 57 44 00 c0 21 44
0040    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080    ...
```

Heartbeat

Sent every second, this packet does not contain any payload.

Heartbeat packet from iPad

```
0000    00 00 00 00 02 00 1e 00 00 00 00 00 00 00 00 00
0010    4d d8 1a 41 00 00 00 00 00 00 20 41 86 c9 e2 36
0020    00 00 00 00 80 88 44 4b 00 00 00 00 00 00 00 00
0030    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
0070   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

6.3. Time Synchronization

Time synchronization takes place on UDP ports 7010 (client) and 7011 (server), using the **NTP** protocol (RFC 5905). The AirPlay server runs an NTP client. Requests are sent to the AirPlay client at 3 second intervals. The reference date for the timestamps is the beginning of the mirroring session.

server → client

```
0000   23 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020   00 00 00 00 00 00 00 00 00 00 01 c4 c8 ac 5d b5


Network Time Protocol
    Flags: 0x23
        00.. .... = Leap Indicator: no warning (0)
        ..10 0... = Version number: NTP Version 4 (4)
        .... .011 = Mode: client (3)
    Peer Clock Stratum: unspecified or invalid (0)
    Peer Polling Interval: invalid (0)
    Peer Clock Precision: 1.000000 sec
    Root Delay: 0.0000 sec
    Root Dispersion: 0.0000 sec
    Reference ID: NULL
    Reference Timestamp: Jan 1, 1970 00:00:00.000000000
UTC
    Origin Timestamp: Jan 1, 1970 00:00:00.000000000 UTC
    Receive Timestamp: Jan 1, 1970 00:00:00.000000000
UTC
    Transmit Timestamp: Jan 1, 1900 00:07:32.783880000
UTC
```

client → server

```
0000   24 01 02 e8 00 00 00 00 00 00 00 00 41 49 52 50
0010   00 00 00 00 00 00 00 00 00 00 01 c4 c8 ac 5d b5
0020   00 00 01 c4 c9 6a 0b a1 00 00 01 c4 c9 78 73 d2

Network Time Protocol
    Flags: 0x24
        00.. .... = Leap Indicator: no warning (0)
        ..10 0... = Version number: NTP Version 4 (4)
        .... .100 = Mode: server (4)
    Peer Clock Stratum: primary reference (1)
    Peer Polling Interval: invalid (2)
    Peer Clock Precision: 0.000000 sec
    Root Delay: 0.0000 sec
    Root Dispersion: 0.0000 sec
    Reference ID: Unidentified reference source 'AIRP'
    Reference Timestamp: Jan 1, 1970 00:00:00.000000000
UTC
    Origin Timestamp: Jan 1, 1900 00:07:32.783880000 UTC
    Receive Timestamp: Jan 1, 1900 00:07:32.786774000
UTC
    Transmit Timestamp: Jan 1, 1900 00:07:32.786994000
UTC
```

7. Password Protection

An AirPlay server can require a password for displaying any content from the network. This is implemented using standard **HTTP Digest Authentication** (RFC 2617), over RTSP for AirTunes, and HTTP for everything else. The digest realms and usernames accepted by Apple TV are the following:

service	realm	username
AirTunes	`raop`	`iTunes`
AirPlay	`AirPlay`	`AirPlay`

Example 1: AirTunes password request

client → server

```
ANNOUNCE rtsp://fe80::217:f2ff:fe0f:e0f6/3414156527
RTSP/1.0
CSeq: 3
Content-Type: application/sdp
Content-Length: 348
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Client-Instance: 56B29BB6CB904862
DACP-ID: 56B29BB6CB904862
Active-Remote: 448488758

<SDP DATA>
```

server → client

```
RTSP/1.0 401 Unauthorized
Server: AirTunes/130.14
WWW-Authenticate: Digest realm="raop",
nonce="ddfd59b4aea7bbbcbbb3b60d3b2768b7"
CSeq: 3
```

client → server

```
ANNOUNCE rtsp://fe80::217:f2ff:fe0f:e0f6/3414156527
RTSP/1.0
CSeq: 4
Content-Type: application/sdp
Content-Length: 348
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Client-Instance: 56B29BB6CB904862
DACP-ID: 56B29BB6CB904862
Active-Remote: 448488758
Authorization: Digest username="iTunes", realm="raop",
nonce="ddfd59b4aea7bbbcbbb3b60d3b2768b7",
uri="rtsp://fe80::217:f2ff:fe0f:e0f6/3414156527",
response="36f93a97c9038598290729ec0f141b03"

<SDP DATA>
```

server → client

```
RTSP/1.0 200 OK
Server: AirTunes/130.14
CSeq: 4
```

Example 2: AirPlay password request

client → server

```
POST /play HTTP/1.1
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Content-Length: 163
Content-Type: text/parameters

Content-Location:
http://192.168.1.18:3689/airplay.mp4?database-spec='d
map.persistentid:0x63b5e5c0c201542e'&item-spec='dmap.
itemid:0x21e'
Start-Position: 0.317546
```

server → client

```
HTTP/1.1 401 Unauthorized
Date: Fri, 09 Mar 2012 15:50:40 GMT
Content-Length: 0
WWW-Authenticate: Digest realm="AirPlay",
nonce="MTMzMTMwODI0MCDEJP5Jo7HFo81rbAcKNKw2"
```

client → server

```
POST /play HTTP/1.1
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3)
AppleWebKit/535.18.5
Content-Length: 163
Content-Type: text/parameters
```

```
Authorization: Digest username="AirPlay",
realm="AirPlay",
nonce="MTMzMTMwODI0MCDEJP5Jo7HFo81rbAcKNKw2",
uri="/play",
response="aa085eea3e66a2e56125a4957e70894a"

Content-Location:
http://192.168.1.18:3689/airplay.mp4?database-spec='d
map.persistentid:0x63b5e5c0c201542e'&item-spec='dmap.
itemid:0x21e'
Start-Position: 0.317546
```

server → client

```
HTTP/1.1 200 OK
Date: Fri, 09 Mar 2012 15:50:40 GMT
Content-Length: 0
```

8. History

Date	Changes
2012-03-20	Initial version.

9. Resources

9.1. IETF RFCs

- RFC 2616: Hypertext Transfer Protocol – HTTP/1.1
- RFC 2617: HTTP Authentication: Basic and Digest Access Authentication
- RFC 2326: Real Time Streaming Protocol (RTSP)
- RFC 4566: SDP: Session Description Protocol
- RFC 3550: RTP: A Transport Protocol for Real-Time Applications
- RFC 5905: Network Time Protocol Version 4
- RFC 4648: The Base16, Base32, and Base64 Data Encodings

9.2. IETF drafts

- Multicast DNS
- DNS-Based Service Discovery
- Reverse HTTP
- HTTP Live Streaming

9.3. Apple Protocols

- DAAP: Digital Audio Access Protocol
- DACP: Digital Audio Control Protocol
- RAOP: Remote Audio Output Protocol