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*, *PTP* 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.

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 the *Airplay* for photo and video and in later versions also audio content and the other one is [RAOP], used for audio streaming and in later versions superseded by *Airplay* service.

_airplay._tcp

```
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	TYPE	DESCRIPTION
model	string	device model
manufacturer	string	device manufacturer
serialNumber	string	device serial number
fv	string	device firmware version
osvers	string	device OS version
deviceid	string	Device ID. Usually MAC address of the device
features	32 bit hex number,optional high order 32 bit hex number	bitfield of supported features. This was originally a 32 bit value but it has since been expanded to a 64 bit value. To support both these types the mDNS value is encoded as two 32 bit values separated by comma with the comma and second 32 bit value being optional.
pw	boolean	server is password protected
acl	int64	Access control level
srcvers	string	airplay version
flags	20 bit hex number	bitfield of status flags
pk	hex string	public key
pi	UUID string	group_id / PublicCUAirPlayPairingIdentifier
psi	UUID string	PublicCUSystemPairingIdentifier
gid	UUID string	group UUID
gcgl	boolean	group contains group leader / Group contains discoverable leader
igl	boolean	is group leader
gpn	string	group public name
hgid	UUID string	home group UUID
hmid	string	household ID
pgcgl	boolean	parent group contains discoverable leader
pgid	UUID string	parent group UUID
tsid	UUID string	3008B5C8-9BD3-4479-A564-90BFB3D780C0
rsf	64 bit hex number	required sender features
protovers	string	protocol version
vv	?	vodka version

_raop._tcp

```
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

CN	DESCRIPTION
3	AAC ELD (Enhanced Low Delay)

Encryption/Authentication Types

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

Metadata Types

MD	BIT	DESCRIPTION
0	17	text
1	15	artwork
2	16	progress
	50	bplist

Subtype

```
AppleTV = deviceModel has prefix “AppleTV”
HomePod = deviceModel has prefix “AudioAccessory”
ThirdPartySpeaker = HasUnifiedAdvertiserInfo or SupportsUnifiedPairSetupAndMFi
feature is set
Unknown = otherwise
```

CanBeRemoteControlled

`SupportsBufferedAudio` is set and `PINRequired` is not set

State Changes

Depending on the state of the device the mDNS record is changed to reflect this. Primarily it is the `flags`, `gid`, `igl`, `gcgl`, `pgid` and `pgcgl` fields that are changed.

Apple TV not playing

```
flags=0x10644
gid=712F0759-5D44-41E7-AB67-FAB0AD39E165
igl=1
gcgl=1
```

Apple TV receiving AirPlay audio

```
flags=0x30e44
gid=19F5D4B2-8A06-4792-923E-8AFA83913238
igl=0
gcgl=0
pgid=19F5D4B2-8A06-4792-923E-8AFA83913238
pgcgl=0
```

Apple TV receiving AirPlay video

```
flags=0x30e44
gid=19F5D4B2-8A06-4792-923E-8AFA83913238
igl=0
gcgl=0
pgid=19F5D4B2-8A06-4792-923E-8AFA83913238
pgcgl=0
```

Apple TV receiving AirPlay screen

```
flags=0x30644
gid=712F0759-5D44-41E7-AB67-FAB0AD39E165
igl=1
gcgl=1
```

Apple TV playing local media

```
flags=0x10644
gid=FA69F5A2-5574-4E4D-A676-CA7F61A904A3
igl=1
gcgl=1
```

Features

EXAMPLE:
Decimal: 0
Hex: 0x0
mDNS: 0x0,0x0

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
6		
7	Screen	mirroring supported
8	ScreenRotate	screen rotation supported
9	Audio	audio supported
10		
11	AudioRedundant	audio packet redundancy supported
12	FPSAPv2pt5_AES_GCM	FairPlay secure auth supported
13	PhotoCaching	photo preloading supported
14	Authentication4	Authentication type 4. FairPlay authentication
15	MetadataFeature1	bit 1 of MetadataFeatures. Artwork.
16	MetadataFeature2	bit 2 of MetadataFeatures. Progress.
17	MetadataFeature0	bit 0 of MetadataFeatures. Text.
18	AudioFormat1	support for audio format 1
19	AudioFormat2	support for audio format 2. This bit must be set for AirPlay 2 connection to work
20	AudioFormat3	support for audio format 3. This bit must be set for AirPlay 2 connection to work
21	AudioFormat4	support for audio format 4
22		
23	Authentication1	Authentication type 1. RSA Authentication
24		
25		
26	HasUnifiedAdvertiserInfo	
27	SupportsLegacyPairing	
28		
29		
30	RAOP	RAOP is supported on this port. With this bit set your don't need the AirTunes service
31		
32	IsCarPlay / SupportsVolume	Don’t read key from pk record it is known
33	SupportsAirPlayVideoPlayQueue	
34	SupportsAirPlayFromCloud	
35		
36		
37		
38	SupportsCoreUtilsPairingAndEncryption	SupportsHKPairingAndAccessControl, SupportsSystemPairing and SupportsTransientPairing

		implies SupportsCoreUtilsPairingAndEncryption
39		
40	SupportsBufferedAudio	Bit needed for device to show as supporting multi-room audio
41	SupportsPTP	Bit needed for device to show as supporting multi-room audio
42	SupportsScreenMultiCodec	
43	SupportsSystemPairing	
44		
45		
46	SupportsHKPairingAndAccessControl	
47		
48	SupportsTransientPairing	SupportsSystemPairing implies SupportsTransientPairing
49		
50	MetadataFeature4	bit 4 of MetadataFeatures. binary plist.
51	SupportsUnifiedPairSetupAndMFi	Authentication type 8. MFi authentication
52	SupportsSetPeersExtendedMessage	

Status Flags

Example:

Decimal: 0

Hex: 0x0

BIT	NAME	DESCRIPTION
0	Problem has been detected	Defined in CarPlay section of MFi spec. Not seen set anywhere
1	Device is not configured	Defined in CarPlay section of MFi spec. Not seen set anywhere
2	Audio cable is attached	Defined in CarPlay section of MFi spec. Seen on AppleTV, Denon AVR, HomePod, Airport Express
3	PINRequired	
4	???	Not seen set anywhere
5	???	Not seen set anywhere
6	SupportsAirPlayFromCloud	
7	PasswordRequired	
8	???	Not seen set anywhere
9	OneTimePairingRequired	
10	DeviceWasSetupForHKAccessControl	
11	DeviceSupportsRelay	Shows in logs as relayable. When set iOS will connect to the device to get currently playing track.
12	SilentPrimary	
13	TightSyncIsGroupLeader	
14	TightSyncBuddyNotReachable	

15	IsAppleMusicSubscriber	Shows in logs as music
16	CloudLibraryIsOn	Shows in logs as iCML
17	ReceiverSessionIsActive	Shows in logs as airplay-receiving. Set when Apple TV is receiving anything via AirPlay.
18	???	Not seen set anywhere
19	???	Not seen set anywhere

Photos

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

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 picture
`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
```

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`

KEY	TYPE	DESCRIPTION
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
```

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
```

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>
```

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.

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	DESCRIPTION
deviceid	string	58:55:CA:1A:E2:88	MAC address
features	integer	14839	0x39f7
model	string	AppleTV2,1	device model

KEY	TYPE	VALUE	DESCRIPTION
protovers	string	1.0	protocol version
srcvers	string	120.2	server version

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='dmap.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 seconds

KEY	TYPE	DESCRIPTION
position	real	playback position in seconds
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 ranges
seekableTimeRanges	array	array of seekable time ranges

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>
```

```
<?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>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/bipbop/gear1/prog_index.m3
    </dict>
  </array>
</dict>
</plist>
```

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
```

Audio

Audio streaming is supported using the [RTSP] protocol.

RTSP Requests

GET /info

KEY	TYPE	VALUE	DESCRIPTION
PTPInfo	string		
build	string		
deviceID	string	58:55:CA:1A:E2:88	MAC address
features	integer	14839	features bits as decimal

KEY	TYPE	VALUE	DESCRIPTION
			value
initialVolume	real		
macAddress	string		
firmwareBuildDate	string		
firmwareRevision	string		
keepAliveLowPower	boolean		
keepAliveSendStatsAsBody	boolean		
manufacturer	string		
model	string	AppleTV2,1	device model
name	string		
nameIsFactoryDefault	boolean		
pi	string		
pk	data		
playbackCapabilities.supportsFPSSecureStop	boolean		
playbackCapabilities.supportsUIForAudioOnlyContent	boolean		
protocolVersion	string	1.0	protocol version
psi	string		
senderAddress	string		
sdk	string		
sourceVersion	string	120.2	server version
statusFlags	integer	4	status flags as decimal value
txtAirPlay	data	...	raw TXT record from AirPlay service mDNS record
txtRAOP	data	...	raw TXT record from AirTunes service mDNS record
volumeControlType	integer		
vv	integer		

CLIENT → SERVER

```
GET /info RTSP/1.0
X-Apple-ProtocolVersion: 1
CSeq: 0
DACP-ID: ADA239F4521B1802
Active-Remote: 753030410
User-Agent: AirPlay/380.10.1
```

SERVER → CLIENT

```
RTSP/1.0 200 OK
Content-Length: 927
Content-Type: application/x-apple-binary-plist
Server: AirTunes/366.0
CSeq: 0

<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>PTPInfo</key>
    <string>OpenAVNU ArtAndLogic-aPTP-changes 1.0</string>
    <key>build</key>
    <string>17.0</string>
    <key>deviceID</key>
    <string>00:05:CD:D4:42:96</string>
    <key>features</key>
    <integer>496155769145856</integer>
    <key>firmwareBuildDate</key>
    <string>Jan 30 2019</string>
    <key>firmwareRevision</key>
    <string>1.505.130</string>
    <key>keepAliveLowPower</key>
    <true/>
    <key>keepAliveSendStatsAsBody</key>
    <true/>
    <key>manufacturer</key>
    <string>Sound United</string>
    <key>model</key>
    <string>AVR-X3500H</string>
    <key>name</key>
    <string>Daemon</string>
    <key>nameIsFactoryDefault</key>
    <false/>
    <key>pi</key>
    <string>650c20ee-842c-4c01-80ed-22b7fb239881</string>
    <key>protocolVersion</key>
    <string>1.1</string>
    <key>sdk</key>
    <string>AirPlay;2.0.2</string>
    <key>sourceVersion</key>
    <string>366.0</string>
    <key>statusFlags</key>
    <integer>4</integer>
    <key>txtAirPlay</key>
    <data>
    BWFjbD0wGmRldmljZWlkPTAwOjA1OkNEOkQ0OjQyOjk2G2ZlYXR1cmVzPTB4NDQ1RjhB
    MDAsMHgxQzM0MAdyc2Y9MHgwEGZ2PXAyMC4xLjUwNS4xMzAJZmxhZ3M9MHg0EG1vZGVs
    PUFWUi1YMzUwMEgZbWFudWZhY3R1cmVyPVNvdW5kIFVuaXRlZBtzZXJpYWxOdW1iZXI9
    QkJXMzYxODEyMTIzNjQNcHJvdG92ZXJzPTEuMQ1zcmN2ZXJzPTM2Ni4wJ3BpPTY1MGMy
    MGVlLTg0MmMtNGMwMS04MGVkLTIyYjdmYjIzOTg4MShnaWQ9NjUwYzIwZWUtODQyYy00
    YzAxLTgwZWQtMjJiN2ZiMjM5ODgxBmdjZ2w9MENwaz0wYWNkN2Q2MWIyODRjMGFmNzFj
    N2VmNGY3ZWE2NDRkZGRlYzIzOGVmMjdjM2MwYWQzZDVkM2JiOWM4YjMxZThm
    </data>
</dict>
</plist>
```

```
YnBsaXN0MDDfEBIBAgMEBQYHCAkKCwwNDg8QERITFBUWFxgZGhcbHB0eHyAh
IiNTc2RrUnBpXxAQZmlybXdhcmVSZXZpc2lvblxtYW51ZmFjdHVyZXJfEBFr
ZWVwQWxpdmVMb3dQb3dlcl8QEWZpcm13YXJlQnVpbGREYXRlVW1vZGVsXxAU
bmFtZUlzRmFjdG9yeURlZmF1bHRfEBhrZWVwQWxpdmVTZW5kU3RhdHNBc0Jv
ZHlbc3RhdHVzRmxhZ3NYZGV2aWNlSURVYnVpbGRadHh0QWlyUGxheVdQVFBJ
bmZvXxAPcHJvdG9jb2xWZXJzaW9uXXNvdXJjZVZlcnNpb25YZmVhdHVyZXNU
bmFtZV1BaXJQbGF5OzIuMC4yXxAkNjUwYzIwZWUtODQyYy00YzAxLTgwZWQt
MjJiN2ZiMjM5ODgxWTEuNTA1LjEzMFxTb3VuZCBVbml0ZWQJW0phbiAzMCAy
MDE5WkFWUi1YMzUwMEgIEARfEBEwMDowNTpDRDpENDo0Mjo5NlQxNy4wTxEB
XwVhY2w9MBpkZXZpY2VpZD0wMDowNTpDRDpENDo0Mjo5NhtmZWF0dXJlcz0w
eDQ0NUY4QTAwLDB4MUMzNDAHcnNmPTB4MBBmdj1wMjAuMS41MDUuMTMwCWZs
YWdzPTB4NBBtb2RlbD1BVlItWDM1MDBIGW1hbnVmYWN0dXJlcj1Tb3VuZCBV
bml0ZWQbc2VyaWFsTnVtYmVyPUJCVzM2MTgxMjEyMzY0DXByb3RvdmVycz0x
LjENc3JjdmVycz0zNjYuMCdwaT02NTBjMjBlZS04NDJjLTRjMDEtODBlZC0y
MmI3ZmIyMzk4ODEoZ2lkPTY1MGMyMGVlLTg0MmMtNGMwMS04MGVkLTIyYjdm
YjIzOTg4MQZnY2dsPTBDcGs9MGFjZDdkNjFiMjg0YzBhZjcxYzdlZjRmN2Vh
NjQ0ZGRkZWMyMzhlZjI3YzNjMGFkM2Q1ZDNiYjljOGIzMWU4Zl8QJU9wZW5B
Vk5VIEFydEFuZExvZ2ljLWFQVFAtY2hhbmdlcyAxLjBTMS4xVTM2Ni4wEwAB
w0BEX4oAVkRhZW1vbgAIAC8AMwA2AEkAVgBqAH4AhACbALYAwgDLANEA3ADk
APYBBAENARIBIAFHAVEBXgFfAWsBdgF3AXkBjQGSAvUDHQMhAycDMAAAAAAA
AAIBAAAAAAAAACQAAAAAAAAAAAAAAAAAAM3
```

POST /pair-setup

POST /pair-verify

POST /fp-setup

POST /auth-setup

In case MFi authentication is supported (HasUnifiedAdvertiserInfo), it can be raised either with a auth-setup challenge or included in the pairing process if supported (SupportsUnifiedPairSetupAndMFi). This section describes the first option.

Note that even if it is a server authentication (so that clients ensure MFi authenticity), with Airplay2 devices this step cannot be ignored from a client implementation point of view. Meaning, even if the authentication/signature is not checked on client side, the request has to be done, otherwise the server will deny further requests

The challenge process is the following:

1. Generation of Curve25119 key pairs on both client and server
2. Client send its public key to server
3. Server append its public key with client's one, this will be the message to sign. It then gets the signature from Apple authentication IC. RSA-1024 is used, with SHA-1 hash algorithm.
4. Signature is encrypted with AES-128 in Counter mode with:

- Key = 16 first bytes of `SHA1(<7:AES-KEY><32:Curve25119 Shared key>)`
- IV = 16 first bytes of `SHA1(<6:AES-IV><32:Curve25119 Shared key>)`

5. Server respond with it's Curve25119 public key, the encrypted signature and the certificate.

A pair-setup request body has the following format:

```
<1:Encryption Type>
<32:Client's Curve25119 public key>
```

Where encryption type is:

VALUE	TYPE
0x00	Invalid
0x01	Unencrypted
0x10	MFi-SAP-encrypted AES key.

A pair-setup response body has the following format:

```
<32:Server's Curve25119 public key>
<4:Certificate length (int32be)>
<n:PKCS#7 DER encoded MFiCertificate>
<4:Signature length (int32be)>
<n:Signature>
```

CLIENT → SERVER

```
POST /auth-setup RTSP/1.0
Content-Type: application/octet-stream
Content-Length: 33
CSeq: 1
User-Agent: iTunes/7.6.2 (Windows; N;)
Client-Instance: 050748FD6F0853AC
DACP-ID: 124D322761E2EC16
Client-instance-identifier: 5743a6b4-e835-412f-9894-14011e386cf2

<BINARY_REQUEST>
```

SERVER → CLIENT

```
RTSP/1.0 200 OK
Content-Length: 1076
Content-Type: application/octet-stream
Server: AirTunes/366.0
CSeq: 1

<BINARY_RESPONSE>
```

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]. 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/n4L5KZkE2AAAAAQlDx69reTdwHF9LaNm
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+lbjp7EhgAAAAQeX5uqGyYkBmJX+gd5
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+686xfviEphhw8AAAAQE/3LSqv9MHgnEKxkb
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;control_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;control_port=63379;timing_
Session: 1
Audio-Jack-Status: connected
Server: AirTunes/130.14
CSeq: 4
```

SETPEERS

POST /command

POST /feedback

POST /audioMode

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
```

FLUSHBUFFERED

Applies to: Airplay2

The `FLUSHBUFFERED` is used to request the flush of the buffer, in a buffered audio usage context (see SupportsBufferedAudio feature)

CLIENT → SERVER

```
FLUSHBUFFERED rtsp://192.168.128.227/4361780817803627124 RTSP/1.0
Content-Length: 87
Content-Type: application/x-apple-binary-plist
CSeq: 12
DACP-ID: C0EF6588CF6D70AC
Active-Remote: 1641836738
User-Agent: AirPlay/387.2

<binary plist>
```

SERVER → CLIENT

```
RTSP/1.0 200 OK
Server: AirTunes/366.0
CSeq: 12
```

binary plist includes the RTP sequence & timestamp to flush up to:

```
{'flushUntilSeq': 15363674, 'flushUntilTS': 239565966}
```

and optionally, the RTP sequence & timestamp to flush from:

```
{'flushFromSeq': 15380707,
 'flushFromTS': 1554930586,
 'flushUntilSeq': 15381800,
 'flushUntilTS': 1556050842}
```

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
```

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: 00000000000000000000000083c117ccafba9b32
```

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
```

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
```

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.

MD	BIT	DESCRIPTION
0	17	text
1	15	artwork
2	16	progress
	50	bplist

Track Informations

Informations about the current track are sent in the [DAAP] format, with `application/x-dmap-tagged` content type.

The following DAAP attributes are displayed on Apple TV:

ATTRIBUTES	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)`

Example: 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
```

AirPort Express Authentication

Sending audio data to the AirPort Express requires a *RSA* based authentication. All binary data are encoded using [Base64] 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:5QYIqmdZGTONY5SHjEJrqAhaa0W9wzDC5i6q221mdGZJ5ubO6Kg
      yhC6U83wpY87TFdPRdfPQl2kVC7+Uefmx1bXdIUo07ZcJsqMbgtje4w2JQw0b
      Uw2BlzNPmVGQOxfdpGc3LXZzNE0jI1D4conUEiW6rrzikXBhk7Y/i2naw13ayy
      xaSwtkiJ0ltBQGYGErbV2tx43QSNj7O0JIG9GrF2GZZ6/UHo4VH+ZXgQ4NZvP/
      QXPCsLutZsvusFDzIEq7TN1fveINOiwrzlN+bckEixvhXlvoQTWE2tjbmQYhMvO
      FIly5gNbZiXi0l5AdolX4jDC2vndFHqWDks/3sPikNg
a=aesiv:zcZmAZtqh7uGcEwPXk0QeA
```

SERVER → CLIENT

```
RTSP/1.0 200 OK
CSeq: 1
Apple-Response: u+msU8Cc7KBrVPjI/Ir8fOL8+C5D3Jsw1+acaW3MNTndrTQAeb/a
          5m10UVBX6wb/DYQGY+b28ksSwBjN0nFOk4Y2cODEf83FAh7B
          mkLpmpkpplp7zVXQ+Z9DcB6gC60ZsS3t98aoR7tSzVLKZNgi2X2sC+vGsz
          utQxX03HK008VjcdngHv3g1p2knoETd07T6eVfZCmPqp6Ga7Dj8VIIj/GEP3
          AjjDx3lJnQBXUDmxM484YXLXZjWFXCiY8GJt6whjf7/2c3rIoT3Z7PQpEvPmM
          1MXU9cv4NL59Y/q0OAVQ38foOz7eGAhfvjOsCnHU25aik7/7ToIYt1tyVtap/kA
Audio-Jack-Status: connected; type=analog
```

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]. 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
```

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.

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:D2:80:0B:68)
sessionID	integer	–808788724	session ID (0xcfcadd0c)
version	string	130.16	server version
param1	data	(72 bytes)	AES key, encrypted with FairPlay
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+AAAAAQukdPJ5Jw/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>
```

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

SIZE	VALUE	DESCRIPTION
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
```

Time Synchronization

Time synchronization takes place on UDP ports 7010 (client) and 7011 (server), using the [NTP] protocol. 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
```

Pairing

In all newer versions of the protocol pairing is required.

HomeKit Based Pairings

Default code is `3939`.

Transient Pairing

`POST /pair-setup` `EncryptionKey` is `SRP shared secret`

Normal Pairing

`POST /pair-setup` `POST /pair-verify`

`EncryptionKey` is agreed `SessionKey`

Encryption

After successful pairing the connection switches to being encrypted using the format `N:n_bytes:tag` where `N` is a 16 bit Little Endian length that describes the number of bytes

in `n_bytes` and `n_bytes` is encrypted using ChaCha20-Poly1305 with `tag` being the Poly1305 tag.

Each direction uses its own key and nonce.

The key for data sent from client to accessory is a HKDF-SHA-512 with the following parameters:

```
InputKey = <EncryptionKey>
Salt = ”Control-Salt”
Info = ”Control-Write-Encryption-Key”
OutputSize = 32 bytes
```

While the data sent from accessory to client is HKDF-SHA-512 with the following parameters:

```
InputKey = <EncryptionKey>
Salt = ”Control-Salt”
Info = ”Control-Read-Encryption-Key”
OutputSize = 32 bytes
```

The nonce is a 64 bit counter (i.e. the high order bits of the full 96 bit nonce is set to 0) starting with 0 and incrementing by 1 for each encrypted block.

MFi Authentication

When SupportsUnifiedPairSetupAndMFi is enabled and HKP is used, the device can authenticate when pairing. In such case M1 step is done with "Pair Setup with Auth" method.

Also, during the M4 step of the pairing process, in addition of the PROOF TLV used in regular pair-setup, the following TLV is added:
`TLV: 0x05,N,ENCRYPTED_DATA_WITH_TAG` where N (int16) is the length of ENCRYPTED_DATA_WITH_TAG

ENCRYPTED_DATA_WITH_TAG has the following format:
`<N:ENCRYPTED_DATA>`
`<16:Poly1305 Tag>`

ENCRYPTED_DATA is encrypted using a HKDF-SHA-512 key with the following parameters:

```
InputKey = <SRP Shared key>
Salt = ”Pair-Setup-Encrypt-Salt”
Info = ”Pair-Setup-Encrypt-Info”
Nonce = ”PS-Msg04”
OutputSize = 32 bytes
```

Decrypted data contains TLVs, which contain MFi Signature (signed by Apple authenticator IC) and used MFi certificate. The message is signed using RSA-1024 with SHA-1 hash

algorithm. The message signed is a HKDF-SHA-512 key with the following parameters:

```
InputKey = <SRP Shared key>
Salt = ”MFi-Pair-Setup-Salt”
Info = ”MFi-Pair-Setup-Info”
OutputSize = 32 bytes
```

Legacy Pairing

TBD

Known Implementations

- https://github.com/mikebrady/shairport-sync
- https://github.com/juhovh/shairplay
- https://github.com/ejurgensen/forked-daapd
- https://github.com/Vluxe/nighthawk
- https://github.com/philippe44/RAOP-Player
- https://github.com/funtax/AirPlayAuth
- https://github.com/espes/Slave-in-the-Magic-Mirror
- https://github.com/robertoandrade/playfair
- https://github.com/phonegapX/AirPlay

https://htmlpreview.github.io/?https://github.com/philippe44/RAOP-Player/blob/master/doc/auth_protocol.html

Contributing

Device Data

If anybody wants to help out who has some Airplay compatible hardware we can always use some more data. Primarily we right now need mDNS records and the results from RTSP `GET /info/` requests. If you have a computer other than a windows machine I have written some short guides below and if that is not enough I can probably help you if you send an e-mail to brian@maven-group.org.

We right now have data for a couple of 4K Apple TVs, an AirPort Express, two stereo paired HomePods, the Libretone LTH200 and a Denon AVR-X3500X surround receiver so if your hardware is something else I am especially interested in the data from it.

Any data that you send will only be viewed by me and I will change stuff like device names, ip addresses, MAC addresses, serial numbers and UUIDs before including the data in this spec.

Linux

If you are on Linux and have `avahi` installed you can run the following shell script to gather all the data.

```
#!/bin/bash
avahi-browse -prt _airplay._tcp | awk -v "FS=;" '{ print $7 }' | sed '/^\s*$/d'
> devices.txt
avahi-browse -aprt | grep -f devices.txt > mDNS.txt
for host in $(cat devices.txt) ; do
PORT="$(avahi-browse -prt _airplay._tcp | grep ";$host;" | awk -v "FS=;" '{print
$9}' | sed '/^\s*$/d')"
curl https://openairplay.github.io/airplay-spec/data/RTSP-get-info-req.bin | nc
-w 2 "$host" $PORT > "RTSP-get-info-res-$host.bin"
done
tar czvf device-data.tar.gz mDNS.txt RTSP-get-info-res*
```

Then you can send the resulting file `device-data.tar.gz` as an e-mail to brian@maven-group.org

macOS

To find mDNS records on macOS I usually use the free Discovery. From there you can find all the devices that publish a `_airplay._tcp` service, select and copy/paste the results into an e-mail. It is also relevant if you can find what other services those devices publish and include the service records for those services.

To get the RTSP `GET /info` results you will need the ip address and port number your devices use for `_airplay._tcp` and in the command below replace `[ip address]`, `[port]` and `[device name]` with the correct values.

```
curl https://openairplay.github.io/airplay-spec/data/RTSP-get-info-req.bin | nc
-w 2 [ip address] [port] > RTSP-get-info-res-[device name].bin
```

Then you can send the mDNS records and `RTSP-get-info-res-[device name].bin` files in an e-mail to brian@maven-group.org

History

Date	Changes
2012–03–20	Initial version.
2019–04–30	Updated with data from reverse engineering `AirPlaySender.framework`.
2019–05–01	Cleaned up HTML and added a little interactivity.
2019–05–01	Added data for Libretone
2019–05–01	Add contributing section
2019–05–02	Add more example devices and click on flag to change

Date	Changes
2019–12–23	Rewrite as mdBook

Resources

IETF RFCs

- [1] RFC 2616: Hypertext Transfer Protocol – HTTP/1.1
- [HTTPBasic] RFC 2617: HTTP Authentication: Basic and Digest Access Authentication
- [RTSP] RFC 2326: Real Time Streaming Protocol (RTSP)
- [SDP] RFC 4566: SDP: Session Description Protocol
- [RTP] RFC 3550: RTP: A Transport Protocol for Real-Time Applications
- [NTP] RFC 5905: Network Time Protocol Version 4
- [Base64] RFC 4648: The Base16, Base32, and Base64 Data Encodings
- [mDNS] RFC 6762: Multicast DNS
- [DNS-SD] RFC 6763: DNS-Based Service Discovery
- [HLS] RFC 8216: HTTP Live Streaming

IETF drafts

- Reverse HTTP

Apple Protocols

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