VOD Interface part 5j

Session Management

PROPRIETARY NOTICE

This document is the property of SeaChange. All information herein is confidential and commercially sensitive to SeaChange and must not be copied or disclosed to any third party without the prior written consent of SeaChange.

Disclosure is permissible provided always that such a disclosure shall only be to people who are directly involved in activities to which this document relates and who have a "need to know"

LEGAL DISCLAIMER

This specification is provided "as is" and without any warranty or representation of any kind, express or implied. Without limitation, there is no warranty of non-infringement, no warranty of merchantability, and no warranty of fitness for a particular purpose. All warranties are expressly disclaimed. User assumes the full risk of using the specification. In no event shall SeaChange be liable for any actual, direct, indirect, punitive, or consequential damages arising from such use, even if advised of the possibility of such damages.

HISTORY

Version	Status	Date	Author	Description
0.9	Draft	19-04-2011	Jan Verhoeven	Version with new template and corrections after internal review session
1.0	Proposed	21-4-2011	Jan Verhoeven	New review comments moved in. First version for internal SeaChange use only.
1.1	Proposed	26-4-2011	Jan Verhoeven	Revised some texts.
1.2	Proposed	2-5-2011	Jan Verhoeven	Changed Stop → DELETE /Session Specified Playlist format and XSD
1.3	Proposed	5-5-2011	W.R. Dittmer	Updated the REST calls and XSDs
1.4	Proposed	18-8-2011	Ralph Janssen	Added Session End Codes Fixed xml typo in example
1.5	Proposed	28-09-2011	W.R. Dittmer	• XSD updated for Channel element. • State return values updated, Torndown removed, Unauthorized added. • State diagram updated. • Sequence diagram updated. • Session end codes updated. • Bookmarks are NOT set using 5J. Use 5Fv2 interface for this. • Added section to describe when clients cannot send a HTTP DELETE with a body.

Document Statuses

This document can have one of the following statuses:

DRAFT Document is draft.

PROPOSED Document is ready for review.

APPROVED Document is review and okay.

RELEASED Document has been reviewed and released by SeaChange development.

Document Version Numbering

This document has unique version numbers to unique states of the document. The version number is expressed as a *major.minor* number pair. These numbers which start at zero are assigned in increasing order and correspond to new developments of the document. Whereby the minor number is used to convey maintenance type of changes and the major one conveys new document releases that has been gone thru a review-release cycle.

Document Change Procedure

Changes to this specification will result in a new version of this document. When this document has either the DRAFT or, PROPOSED status, change requests must be sent to the author. In case the document has APPROVED and RELEASED status, change requests must be submitted to the *Change Control Board* (CCB) of SeaChange development.

Document Change Notice

SeaChange keeps the right to improve this document without notice.

SeaChange®

Contents

1 Introduction

This document describes the session management interaction between multiscreen clients (STB, phones, PC's), the back-office components Publisher and Session Manager (SM) and a CDN. The specification can be used by both CDN providers and multiscreen client developers.

1.1 References

This paragraph lists the references made in this document:

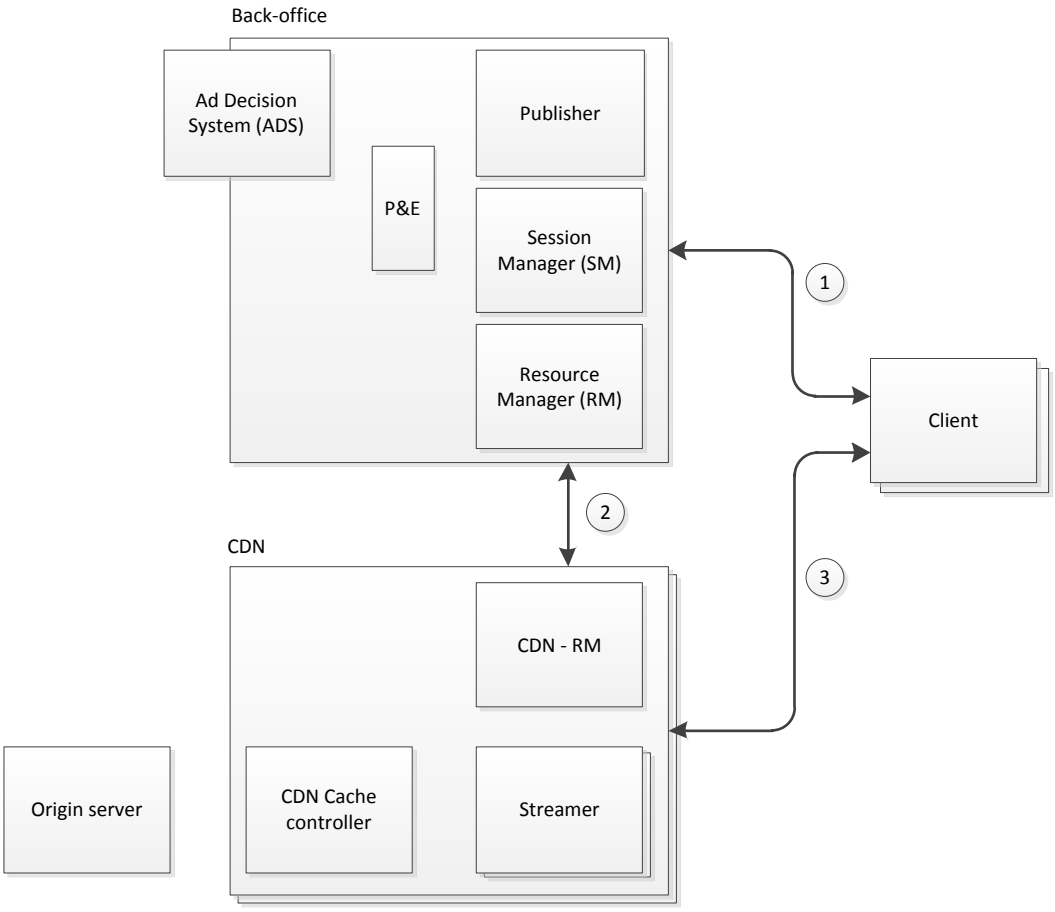
[RFC2326]	Real Time Streaming Protocol (RTSP) – RFC 2326
[IF-5fv2]	Vod Interface part 5f – Traxis.Web
[IF-5i]	Vod Interfaces part 5i – URL signing and validation

2 Context

The intent of this document is to specify the session management interfaces between the following components:

1. The multiscreen client – Back-office
2. The back-office – CDN
3. The multiscreen client - CDN

The following picture shows those interfaces.



Responsibilities of key elements in this picture:

1. Publisher: Provides the client with access to the catalog, assets and products. It deals with purchase and entitlement services.

2. Session Manager (SM): Knows the concept of sessions. Keeps track of the client's player actions such as how long content was viewed, how long it was in pause mode etc. It also keeps track if the session is still active or is torn down.
3. Back-office Resource manager (RM): Balances the different CDN instances in the network and assigns the most relevant CDN (based on load, location, protocol and capabilities) on session setup. Use case: When different CDN's are used for different types of delivery protocols, e.g. HTTP and RTSP.
4. CDN Resource Manager (CDN-RM): Has knowledge of the CDN topology. As such is in charge of forwarding streaming requests to the 'most optimal' streamer.
5. Streamers: Deliver the bytes and manifests (e.g. m3u8 files) to the client device. Typically streams the bytes either from local storage, a cache in the network or the origin server. Caching strategy may be implemented differently by each CDN provider.

This interface document does NOT intend to specify any internals of:

1. The back-office, e.g. how sessions are logged, or how playlists are composed.
2. The CDN, e.g. how edge caching or streaming works.

This interface specification also does not specify how the DRM is handled. It only concentrates on session management.

2.1 Assumptions

The session management interface in this document is based on a number of assumptions described below.

2.1.1 Business models

The following advanced business models related to streaming will be supported on the back-office when the interfaces from this document are implemented on the client and CDN:

1. Allow time based access to the content, e.g. 1 hour of access to all content.
2. Allow no more than X concurrent sessions for any customer or device. Prohibit any new sessions to be created when X concurrent sessions are already active.
3. Deny access to (the not yet downloaded part of) the content in case an entitlement expires.

2.1.2 Sessions

Two session types can be identified:

1. The 'download' session: This is the session that encompasses the timespan during which bytes are retrieved from the CDN.

2. The ‘viewing’ session: This is the session that encompasses the timespan during which the client player plays the content.

A piece of content may be downloaded first and then viewed many times.

The back-office has requirements to report back on viewing sessions, not on download sessions. The download sessions could be considered a responsibility of the CDN. As such all sessions in this document relate to the viewing session.

2.1.3 Deny access to content

In order for the back-office to enforce the above business models, it is necessary that the client can be denied access to the content after the streaming has started. This may be accomplished by two means:

1. Via the player: Inform the player to stop requesting new chunks and the play-out of content.
2. Via the CDN: Inform the CDN to stop serving new chunks to the client in question.

Both options are allowed via the provided Session Management interface using the KeepAlive message. It returns the state in which the Session is. If the state is ‘Torndown’ the client should no longer request new chunks. If the CDN issues the KeepAlive messages and the state is ‘Torndown’ it should reject any new requests for data.

Note that a CDN typically implements no interface with the back-office. Denying access should be possible in those situations as well.

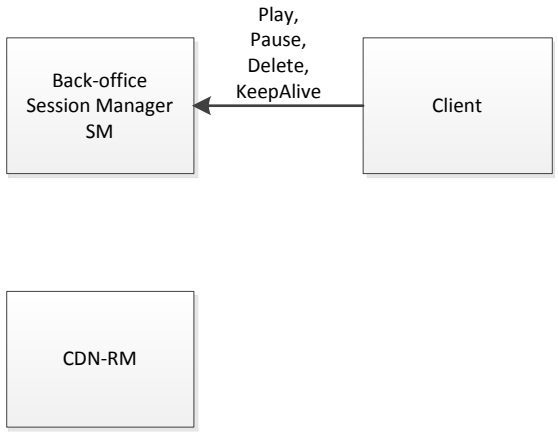
IMPORTANT NOTE: Denying access to the content shall be done properly via a DRM system.

2.1.4 Clients players vs Streamers

The back office may be notified of the state changes in the viewing session either by the client player or the CDN:

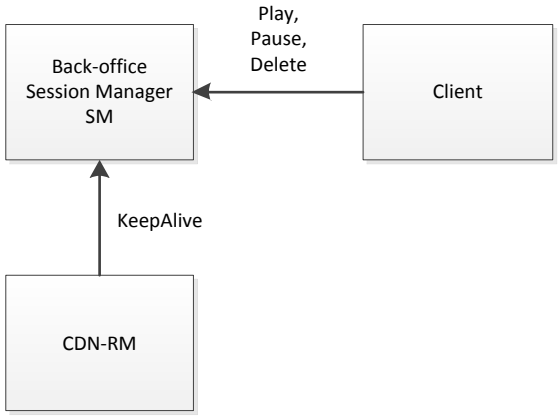
1. All messages from client device: Play, Pause, DELETE, KeepAlive

Option 1: All messages from client



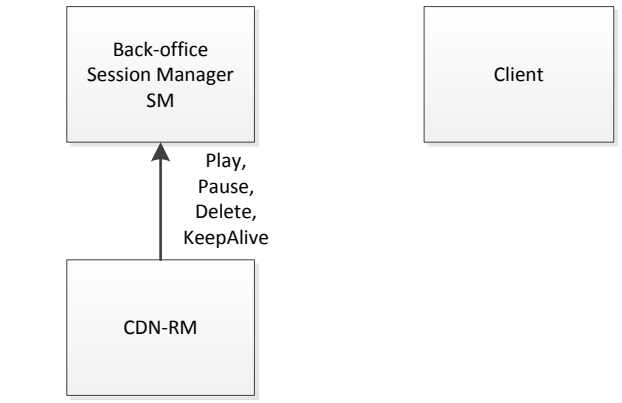
2. Messages from client: Play, Pause, Delete. Messages from streamer: KeepAlive.

Option 2: All messages from client except KeepAlive



3. All messages from streamer device: Play, Pause, Delete, KeepAlive

Option 3: All messages from CDN



Note that option 3 requires a means for the server to detect/predict the session state from the download access pattern (e.g. no new block of data requested for X seconds means Pause).

The KeepAlive in Option 1 may be difficult to implement for many client devices since it requires the availability of a timed callback, e.g. via a separate thread of execution.

2.1.5 Playlist support

A playlist is a list of contents that are played in sequence, back to back. A playlist is typically used in an advertisement model. A playlist is retrieved by the back-office, typically from an Ad Decision System.

Two playlist models are supported in this document:

1. The playlist is provided to the client which is responsible for requesting each of the individual playlist assets.
2. The CDN is given a playlist by the back-office during session setup. The playlist is conveyed via a separate attribute in the URL specified in the 5i interface. This attribute is called 'R' and references a comma delimited list of assets that shall be played in sequence.

Whether model 1 or 2 is used is determined by the capabilities of the CDN/Streamer. Model 2 can only be used if the CDN supports to play all mentioned elements back to back as if they were a single asset.

2.2 Typical streaming session

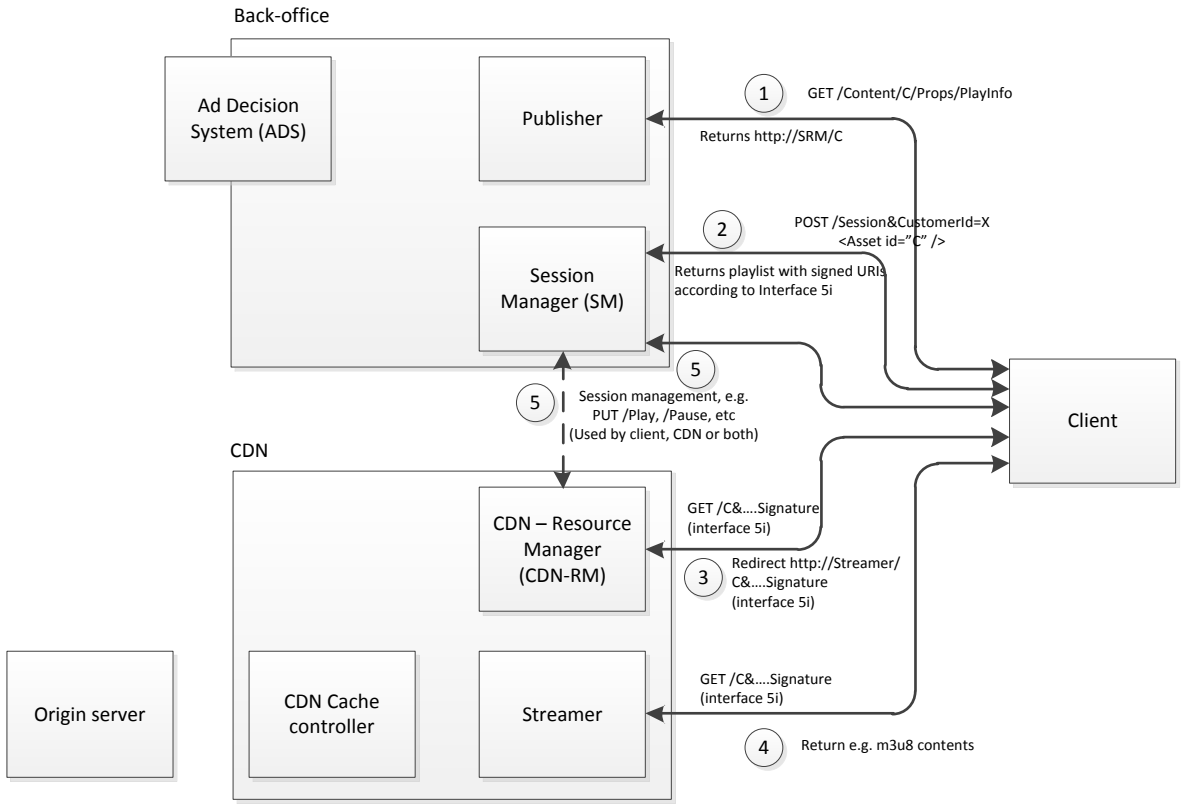
A typical streaming session will go through the following steps:

1. The client requests PlayInfo from Publisher according to the [IF-5fv2] specification. The returned PlayInfo will redirect the client to the Back-office Session Manager.
2. Upon request of the client the Session manager will create a session and returns a playlist file to the client. The playlist file contains a list URIs and their start- and end times and (if needed) trick

masks that indicate whether it is allowed to FF or jump forward (play playlist structure TBD). All URIs pointing to the CDN must be signed according to the [IF-5i] specification.

3. The CDN SRM checks the URI signature to see if the request is authorized. If so, it redirects the request to a streamer that can handle it.
4. The streamer checks and validates the URI signature and sends the manifest file or the requested bytes to the client device.
5. The client informs the session manager about the viewer actions (play, pause, FF etc).

The picture below shows the mentioned steps:



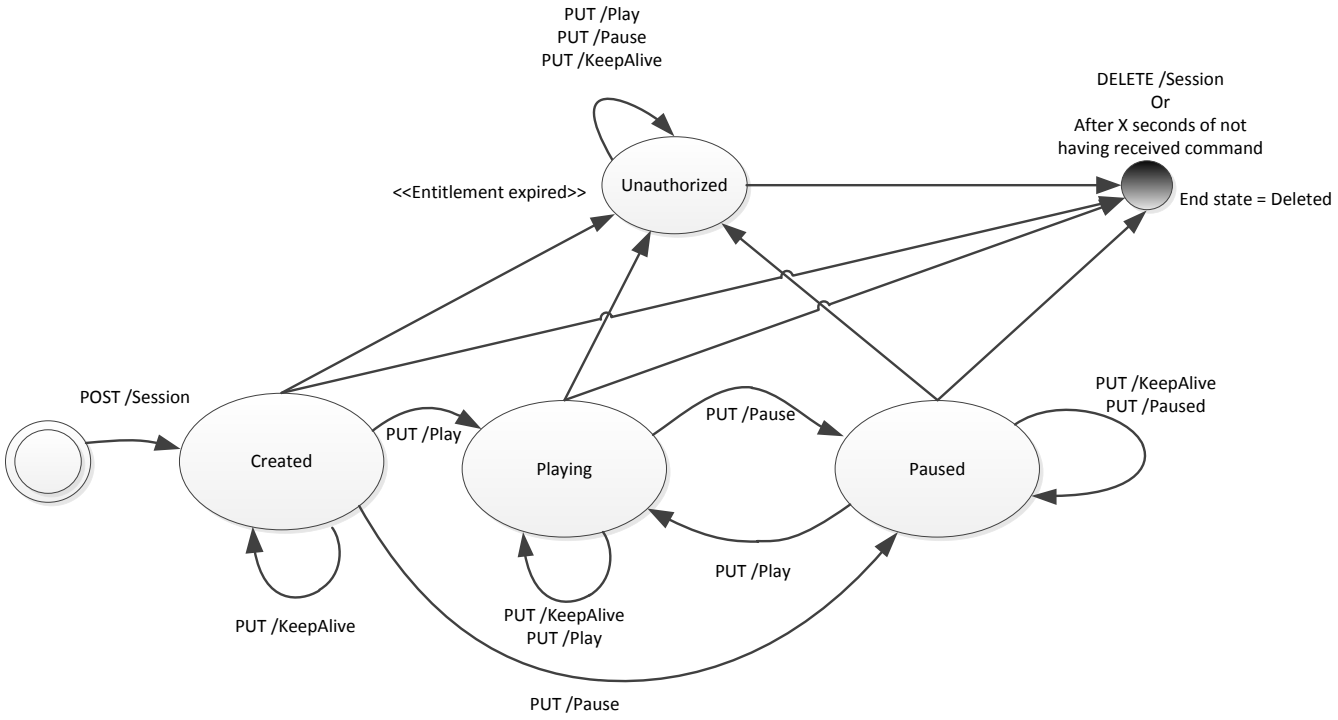
The URI's used by interfaces 2, 3 and 4 are all signed and validated according to interface specification 5i. Interface 1 is according to interface specification 5fv2. Interface 5 is session management interface specified in this document.

2.3 Session management

The following interface describes the session management interface between the client and the Back-office Session manager.

2.3.1 States

The session manager knows the following states:



1. Created: When the session is created by the session manager at the moment the client connects for the first time given the URI to the Session Manager he got from the Publisher.
2. Playing: When the Play command has been issued
3. Paused: When the Pause command has been issued
4. Unauthorized: When the entitlement has expired
5. And the end state when the session is deleted.

See the following section for a detailed description of the commands that must be issued by the client device (or CDN) to signal the appropriate state changes.

3 Interfaces

3.1 Interface binding

The interface binding is based on REST services using HTTP according Appendix C: RESTful services.

An XML structure is optionally used in the standard body of the HTTP PUT or POST requests. On each type of HTTP request an XML structure in the response body can be returned. The response body of the HTTP GET request will always contain an XML structure.

The following semantics are applicable for the interface protocol:

- HTTP/1.1
- text/xml as media type

The communication may also be done in JSON. See [IF-5f] for more information.

3.1.1 Response handling

The response on a HTTP request is an HTTP status codes according to the “Success” or “Client Error” classes (i.e. status codes 2xx, 4xx or 5xx) of status codes.

3.2 Client – back-office session interface

3.2.1 Generic – Time scales

The Play, Pause, Delete and KeepAlive commands all relay the current time offset to the back-office for trick history purposes. The trick history may be used to report back on which parts of the content have actually been viewed. This is especially important for advertisement purposes.

In case of playlists, time is

1. Relative with respect to the entire aggregate playlist
2. Relative with respect to the playlist item which is currently played back.

The <TimeOffset> that is provided shall be relative to that track in the playlist. <TrackId> and <TimeOffset> are mandatory for all messages. In case no playlist is used or in case of an aggregate playlist <TrackId> shall be provided with value 0.

3.2.1.1 NPT

Time shall be provided as an NPT value conform RFC 2326. Only npt-sec and npt-hhmmss are allowed.

4 Appendix A: Usage interfaces

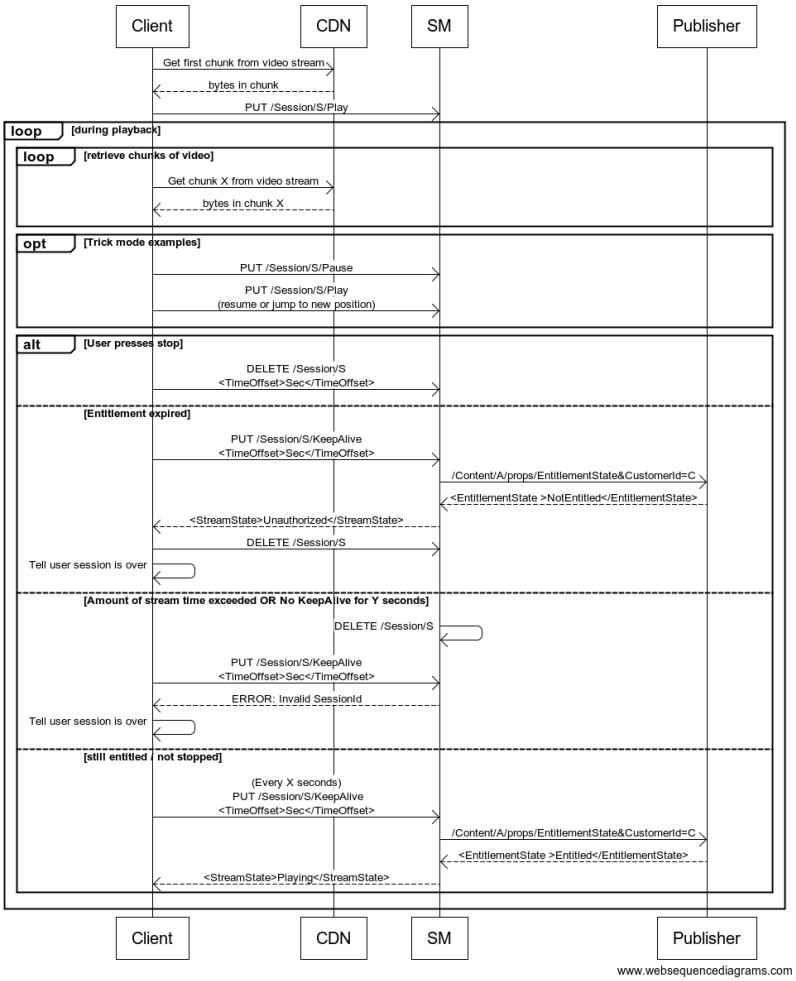
This appendix contains some examples how the content propagation interface is used.

4.1 Generic error

```
HTTP/1.1 404 Not Found
```

4.2 Sequence diagram

The following picture shows the interaction between a typical client, the CDN, the session manager in the back-office and the publisher (or purchase and entitlement entity).



Steps:

1. After the first parts of the video have been retrieved from the CDN, the client device notifies the Session Manager that it has started playing.
2. During playback the following interaction is typical:
    a. The system retrieves one or more chunks of data from the video server
    b. The change of position or speed are signaled via /Play and /Pause commands.
    c. Every X seconds the client or the streaming server signals the back-office that the session is still running via the /KeepAlive command.
    d. A client session can be stopped either by:
        i. The customer pressing the stop button and informing the back-office via an HTTP DELETE command for the session
        ii. An entitlement is expired. The SM finds out with the P&E component in the publisher if this is the case.
        iii. A session has consumed all tokens (time based) or the session manager has not received communication from any client device. The keep-alive will result in an Invalid session id, since the session has already been deleted by the SM.
    e. If the client signals a keepAlive and the customer is still entitled, the state Created, Playing or Paused is returned.

4.3 Session operations Client – Back-office

Session operations are typically done with REST based operations (POST, PUT, GET, DELETE). However, in much the same way as with the 5F interface the GET command can also be used for all commands using the Method attribute. See interface 5F for more details.

4.3.1 Create a session

4.3.1.1 Create a VOD session

By issuing a POST command with the CpeId, if the CpeId is not known the CustomerId must be provided:

```
POST http://localhost/traxis/web/Session/propset/all?CpeId=882 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1
Host: localhost
Content-Length: 183
Content-Type: text/xml

<?xml version="1.0" encoding="utf-8"?>
<CreateSession>
<ContentId>crid:~~2F~~2Fupc.com~~2F1001~~2F00000000000p1,imi:056351329be74425b0b3ca5566
     212620</ContentId>
</CreateSession>
```

The response is:

```
HTTP/1.1 200 OK
Content-Length: 440
Content-Type: text/xml
Server: Traxis 0.0.0.0 Microsoft-HTTPAPI/2.0
Duration: 220739 ms
ResourceInspectionCount: 0
Date: Thu, 05 May 2011 12:32:22 GMT

<?xml version="1.0" encoding="utf-8"?>
<Session id="3e2180f4-ee54-4b66-9130-e52822cbae73" xmlns="urn:eventis:traxisweb:1.0">
  <Playlist id="35a5a4ba-4b8a-421b-978e-28de2ccfea8c">
    <Asset id="a8c2ce91-906e-429e-8611-a4787f777cbc" ordinal="0" allowPause="true"
     allowFastForward="true" allowRewind="true"
     allowSkip="true">http://localhost/a8c2ce91-906e-429e-8611-a4787f777cbc</Asset>
  </Playlist>
  <State>Created</State>
</Session>
```

4.3.1.2 Create a Linear session

To request to start a session of a linear broadcast channel:

```
POST http://localhost/traxis/web/Session/propset/all?CpeId=882 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1
Host: localhost
Content-Length: 153
Content-Type: text/xml

<?xml version="1.0" encoding="utf-8"?>
<CreateSession>
  <ChannelId>service:~~2F~~2F1</ChannelId>
</CreateSession>
```

The response is:

```
HTTP/1.1 200 OK
Content-Length: 325
Content-Type: text/xml
Server: Traxis 0.0.0.0 Microsoft-HTTPAPI/2.0
Duration: 733 ms
ResourceInspectionCount: 0
Date: Thu, 25 May 2011 13:12:02 GMT

<?xml version="1.0" encoding="utf-8"?>
<Session id="3e2180f4-ee54-4b66-9130-e52822cbae73" xmlns="urn:eventis:traxisweb:1.0">
  <Playlist id="35a5a4ba-4b8a-421b-978e-28de2ccfea8c">
    <Channel id="service1">dvb://0001.0F0F.0012</Channel>
  </Playlist>
  <State>Created</State>
</Session>
```

4.3.2 Play

The Play command shall be issued in the following conditions:

1. When the Play button on the client is hit or when the client Player application is started with the content reference and is in auto play mode.
2. When the client scrubs to a new position.
3. When the client fast forwards or rewinds to a new position.

4.3.2.1 Start streaming or Resume after pause

Example when starting playback from position 0:

```
PUT http://localhost/traxis/web/Session/3e2180f4-ee54-4b66-9130-
     e52822cbae73/Play/props/State?CpeId=882 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1
Host: localhost
Content-Length: 134
Content-Type: text/xml

<?xml version="1.0" encoding="utf-8"?>
<PlayNotification>
  <TrackId>0</TrackId>
  <TimeOffset>0</TimeOffset>
</PlayNotification>
```

The response is:

```
HTTP/1.1 200 OK
Content-Length: 163
Content-Type: text/xml
Server: Traxis 0.0.0.0 Microsoft-HTTPAPI/2.0
Duration: 10 ms
ResourceInspectionCount: 0
Date: Thu, 05 May 2011 12:37:05 GMT

<?xml version="1.0" encoding="utf-8"?>
<Session id="3e2180f4-ee54-4b66-9130-e52822cbae73" xmlns="urn:eventis:traxisweb:1.0">
  <State>Playing</State>
</Session>
```

4.3.2.2 Scrubbing/jumping

In case the play command is issued for a scrub action the position is mandatory. Also, the previous position shall be provided so it can be tracked where the customer was before the scrubbing started.

```
PUT http://localhost/traxis/web/Session/3e2180f4-ee54-4b66-9130-
     e52822cbae73/Play/props/State?CpeId=882 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1
Host: localhost
Content-Length: 182
Content-Type: text/xml

<?xml version="1.0" encoding="utf-8"?>
<PlayNotification>
  <TrackId>0</TrackId>
  <TimeOffset>20</TimeOffset>
  <PreviousTimeOffset>10</PreviousTimeOffset>
</PlayNotification>
```

The response is:

```
HTTP/1.1 200 OK
Content-Length: 163
Content-Type: text/xml
Server: Traxis 0.0.0.0 Microsoft-HTTPAPI/2.0
Duration: 0 ms
ResourceInspectionCount: 0
Date: Thu, 05 May 2011 12:38:35 GMT

<?xml version="1.0" encoding="utf-8"?>
<Session id="3e2180f4-ee54-4b66-9130-e52822cbae73" xmlns="urn:eventis:traxisweb:1.0">
  <State>Playing</State>
</Session>
```

Note: While the customer is scrubbing through the file, no updates should be given to the system to avoid flooding of messages.

4.3.2.3 Fast forward/ Rewind

In client devices (like set-top boxes) that use FF/REW to navigate through the files, the scale factor shall be provided. The Scale factor is a double value as specified in [RFC2326], with a positive value for FF and a negative value for REW. For value 0 (pause) the Pause command shall be issued instead of the Play command.

Example where the player switches to 4 speed FF:

```
PUT http://localhost/traxis/web/Session/3e2180f4-ee54-4b66-9130-
     e52822cbae73/Play/props/State?CpeId=882 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1
```

```
Host: localhost
Content-Length: 159
Content-Type: text/xml

<?xml version="1.0" encoding="utf-8"?>
<PlayNotification>
  <TrackId>0</TrackId>
  <TimeOffset>10</TimeOffset>
  <Scale>4.0</Scale>
</PlayNotification>
```

The response is:

```
HTTP/1.1 200 OK
Content-Length: 163
Content-Type: text/xml
Server: Traxis 0.0.0.0 Microsoft-HTTPAPI/2.0
Duration: 1 ms
ResourceInspectionCount: 0
Date: Thu, 05 May 2011 12:40:46 GMT

<?xml version="1.0" encoding="utf-8"?>
<Session id="3e2180f4-ee54-4b66-9130-e52822cbae73" xmlns="urn:eventis:traxisweb:1.0">
  <State>Playing</State>
</Session>
```

4.3.3 Pause

Upon going into pause mode, the current position and TrackId shall be provided.

Example of player going into pause at 10 seconds from start of track 0:

```
PUT http://localhost/traxis/web/Session/3e2180f4-ee54-4b66-9130-
     e52822cbae73/Pause/props/State?CpeId=882 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1
Host: localhost
Content-Length: 140
Content-Type: text/xml

<?xml version="1.0" encoding="utf-8"?>
<PauseNotification>
  <TrackId>0</TrackId>
  <TimeOffset>10</TimeOffset>
</PauseNotification>
```

The response is:

```
HTTP/1.1 200 OK
Content-Length: 162
Content-Type: text/xml
Server: Traxis 0.0.0.0 Microsoft-HTTPAPI/2.0
Duration: 1 ms
ResourceInspectionCount: 0
Date: Thu, 05 May 2011 12:42:03 GMT

<?xml version="1.0" encoding="utf-8"?>
<Session id="3e2180f4-ee54-4b66-9130-e52822cbae73" xmlns="urn:eventis:traxisweb:1.0">
  <State>Paused</State>
</Session>
```

4.3.4 Session ending

Upon ending the current session, the current position shall be provided. Example:

```
DELETE http://localhost/traxis/web/Session/3e2180f4-ee54-4b66-9130-
     e52822cbae73?CpeId=882 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1
Host: localhost
Content-Length: 237
Content-Type: text/xml

<?xml version="1.0" encoding="utf-8"?>
<DeleteSession>
  <TrackId>0</TrackId>
  <TimeOffset>20</TimeOffset>
  <SessionEndCode>23</SessionEndCode>
  <SessionEndDescription>End of stream</SessionEndDescription>
</DeleteSession>
```

The response is:

```
HTTP/1.1 200 OK
Content-Length: 0
Content-Type: text/xml
Server: Traxis 0.0.0.0 Microsoft-HTTPAPI/2.0
Duration: 30 ms
ResourceInspectionCount: 0
Date: Thu, 05 May 2011 12:48:02 GMT
```

Session end codes 0 - 99 are reserved for the back office.
Any other end codes may be defined by the customer in a configuration file, which will be passed through to reporting/billing systems.

If no end code or an unknown (not customer defined and not reserved) is send, this will be mapped to end code 0.

The currently known end codes and their description of the reserved end codes are:

Session End Code	Description
0	The session was ended for an unknown reason
1	The session was successfully delivered to completion
2	The session was terminated by the customer
3	After beginning a stream, the connection with the client timed out or was lost
4	The session was terminated because it received an Abort instruction from the session manager
5	An error on the server caused the stream to fail
6	The server was unable to locate a resource it needed to fulfill the stream request
7	The server encountered a media format that it does not support

The session end description will be passed on as is (if available).

4.3.4.1 Sending a DELETE with body for clients that do not support bodies with DELETE

There are three ways to still send a body with a DELETE.

Either included the body as query parameter with `&Body=<escaped body>`.

```
DELETE http://localhost/traxis/web/Session/3e2180f4-ee54-4b66-9130-
     e52822cbae73?CpeId=882&Body=%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22utf-
     8%22%3F%3E%3CDeleteSession.... HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1
Host: localhost
```

Or alternatively use a PUT or POST with a body and override the method with the query parameter `&Method=DELETE`:

```
PUT http://localhost/traxis/web/Session/3e2180f4-ee54-4b66-9130-
     e52822cbae73?CpeId=882&Method=DELETE HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1
Host: localhost
Content-Length: 237
Content-Type: text/xml

<?xml version="1.0" encoding="utf-8"?>
<DeleteSession>
  <TrackId>0</TrackId>
```

```
  <TimeOffset>20</TimeOffset>
  <SessionEndCode>23</SessionEndCode>
  <SessionEndDescription>End of stream</SessionEndDescription>
</DeleteSession>
```

Of course one may also use a GET and use both the `&Method=` and `&Body=` query parameters.

4.3.5 Session KeepAlive

The KeepAlive serves the following purposes:

1. Tell the back-office the client is still alive and watching and thus can be billed if needed.
2. Find out if the session must be aborted due to all sorts of server side regulations.

KeepAlive returns one of the following states in the <State> element:

1. Created
2. Playing
3. Paused
4. Unauthorized

If the state is 'Unauthorized' the client should stop the session by sending a session end.

Note that if the state becomes 'Unauthorized' that a play or pause will not change the state anymore.
It will remain 'Unauthorized' until deleted.

Also note that if a session has been transferred to another machine, due to either machine failure or the keep-alives being sent by another party, the state will be 'Created' until changed.

4.3.5.1 KeepAlive – Still streaming

In this case the keep alive is sent and the back-office returns that it is still streaming.

```
PUT http://localhost/traxis/web/Session/3e2180f4-ee54-4b66-9130-
     e52822cbae73/KeepAlive/props/State?CpeId=882 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1
Host: localhost
Content-Length: 150
Content-Type: text/xml

<?xml version="1.0" encoding="utf-8"?>
<KeepAliveNotification>
  <TrackId>0</TrackId>
  <TimeOffset>20</TimeOffset>
</KeepAliveNotification>
```

The response is:

```
HTTP/1.1 200 OK
Content-Length: 162
Content-Type: text/xml
Server: Traxis 0.0.0.0 Microsoft-HTTPAPI/2.0
Duration: 4 ms
ResourceInspectionCount: 0
Date: Thu, 05 May 2011 12:43:30 GMT

<?xml version="1.0" encoding="utf-8"?>
<Session id="3e2180f4-ee54-4b66-9130-e52822cbae73" xmlns="urn:eventis:traxisweb:1.0">
  <State>Paused</State>
</Session>
```

4.3.5.2 KeepAlive – No longer entitled

In this case the keep alive is sent and the back-office returns that the client should terminate the session.

```
HTTP/1.1 200 OK
Content-Length: 162
Content-Type: text/xml
Server: Traxis 0.0.0.0 Microsoft-HTTPAPI/2.0
Duration: 4 ms
ResourceInspectionCount: 0
Date: Thu, 05 May 2011 12:43:30 GMT

<?xml version="1.0" encoding="utf-8"?>
<Session id="3e2180f4-ee54-4b66-9130-e52822cbae73" xmlns="urn:eventis:traxisweb:1.0">
  <State>EntitlementTimedOut</State>
</Session>
```

NOTE: there are many more reasons: CustomerDeleted, CpeDeleted, ContentDeleted,ProductDeleted, etc. Maybe a more generic term is required here.

5 Appendix B: REST services XML schemas

The following XSD describes the XML schema for the rest based services:

```
<?xml version="1.0" encoding="utf-8"?>
<schema attributeFormDefault="unqualified" elementFormDefault="qualified"
     targetNamespace="urn:eventis:traxisweb:1.0" xmlns:tns="urn:eventis:traxisweb:1.0"
     xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="CreateSession">
    <complexType>
      <choice>
        <element name="ContentId" type="string" />
        <element name="ChannelId" type="string" />
      </choice>
    </complexType>
  </element>
  <element name="PlayNotification">
    <complexType>
      <sequence>
        <element name="TrackId" type="int" />
        <element name="TimeOffset" type="string" />
        <element name="PreviousTimeOffset" type="string" minOccurs="0" />
        <element name="Scale" type="double" minOccurs="0" />
      </sequence>
    </complexType>
  </element>
  <element name="PauseNotification">
    <complexType>
      <sequence>
        <element name="TrackId" type="int" />
        <element name="TimeOffset" type="string" />
      </sequence>
    </complexType>
  </element>
  <element name="KeepAliveNotification">
    <complexType>
      <sequence>
        <element name="TrackId" type="int" />
        <element name="TimeOffset" type="string" />
      </sequence>
    </complexType>
  </element>
  <element name="DeleteSession">
    <complexType>
      <sequence>
        <element name="TrackId" type="int" />
        <element name="TimeOffset" type="string" />
        <element name="SessionEndCode" type="int" minOccurs="0" />
        <element name="SessionEndDescription" type="string" minOccurs="0" />
      </sequence>
```

```
    </complexType>
  </element>
  <complexType name="SessionType">
    <all>
      <element minOccurs="0" name="Playlist">
        <complexType>
          <sequence>
            <element minOccurs="0" maxOccurs="unbounded" name="Asset">
              <complexType>
                <simpleContent>
                  <extension base="string">
                    <attribute name="id" type="string" use="required" />
                    <attribute name="ordinal" type="int" use="optional" />
                    <attribute name="inPointInNpt" type="string" use="optional" />
                    <attribute name="outPointInNpt" type="string" use="optional" />
                    <attribute name="allowPause" type="boolean" use="optional"
    default="true" />
                    <attribute name="allowFastForward" type="boolean" use="optional"
    default="true" />
                    <attribute name="allowRewind" type="boolean" use="optional"
    default="true" />
                    <attribute name="allowSkip" type="boolean" use="optional"
    default="true" />
                  </extension>
                </simpleContent>
              </complexType>
            </element>
            <element name="Channel" minOccurs="0">
              <complexType>
                <simpleContent>
                  <extension base="string">
                    <attribute name="id" type="string" use="required" />
                  </extension>
                </simpleContent>
              </complexType>
            </element>
          </sequence>
          <attribute name="id" type="string" use="required" />
        </complexType>
      </element>
      <element minOccurs="0" name="State" type="string" />
    </all>
    <attribute name="id" type="string" use="required" />
  </complexType>
  <element name="Session" type="tns:SessionType" />
</schema>
```

6 Appendix C: RESTful services

6.1 Background

Nowadays many systems use XML-based messages to exchange information over computer networks. Several protocols are specified in the past by several organizations to support the exchange of these XML-based messages over HTTP. Some of these protocols are for instance XML-RPC and SOAP. These protocols claim to offer Web services over HTTP. However, in many ways these protocols don't follow the architecture of the Web. SOAP is used for an interoperable cross-platform remote procedure call (RPC) but can be complex to handle between different platforms.

6.2 REST

Now there is REST. REST is an architectural style that uses the strengths of the Web to build services (more detailed information about REST can be found at **Error! Reference source not found.Error! Reference source not found.**). It proposes a set of constraints that simplifies interfaces for integration. Interfaces that are compliant to REST are called RESTful services. REST embraces the Web where each resource is identified by a single URI. RESTful services model the interaction with user agents based on resources. Each such resource is represented by a unique URI, and the user agent uses the uniform interface of HTTP to interact with a resource via that URI.

User agents only interact with resources using the prescribed HTTP verbs. The four main verbs of the uniform HTTP interface are POST, GET, PUT, and DELETE. These verbs are often associated with the CREATE, READ, UPDATE and DELETE (CRUD) operations used in database technologies. The following table provides an overview of these commands:

HTTP	CRUD	Usage
POST	Create	Used to create a new resource in a collection where the host creates the resource (e.g. the user agent does not supply the unique resource). As a result this resource is returned to the user agent.
GET	Read	Used to retrieve a read-only representation of a resource
PUT	Update, Create	Used to modifying an existing resource in the host. In case the provided resource does not exist it will be created in the host (e.g. the user agent supplies the unique resource)
DELETE	Delete	Used to delete a resource in the host

In case of complex operations an XML structure can be used in the request and/or response to provide/retrieve attributes of the unique resource.

As a result of these HTTP commands a response is returned to the user agent using the standard HTTP response status codes.

Diagrams

Regenerate diagrams with www.websequencediagrams.com

```
Client -> CDN: Get first chunk from video stream
CDN --> Client: bytes in chunk
Client -> SM: PUT /Session/S/Play
loop during playback

loop retrieve chunks of video
  Client -> CDN: Get chunk X from video stream
  CDN --> Client: bytes in chunk X
end

opt Trick mode examples
  Client -> SM: PUT /Session/S/Pause
  Client -> SM: PUT /Session/S/Play\n(resume or jump to new position)
end

alt User presses stop
    Client -> SM: DELETE /Session/S\n<TimeOffset>Sec</TimeOffset>

else Entitlement expired
  Client -> SM: PUT /Session/S/KeepAlive\n<TimeOffset>Sec</TimeOffset>
  SM -> Publisher: /Content/A/props/EntitlementState&CustomerId=C
  Publisher --> SM: <EntitlementState >NotEntitled</EntitlementState>
  SM --> Client: <StreamState>Unauthorized</StreamState>
  Client -> SM:DELETE /Session/S
  Client -> Client: Tell user session is over

else Amount of stream time exceeded OR No KeepAlive for Y seconds
    SM -> SM: DELETE /Session/S
    Client -> SM: PUT /Session/S/KeepAlive\n<TimeOffset>Sec</TimeOffset>
    SM --> Client: ERROR: Invalid SessionId
    Client -> Client: Tell user session is over

else still entitled / not stopped
  Client -> SM: (Every X seconds)\n PUT /Session/S/KeepAlive\n<TimeOffset>Sec</TimeOffset>
  SM -> Publisher: /Content/A/props/EntitlementState&CustomerId=C
  Publisher --> SM: <EntitlementState >Entitled</EntitlementState>
  SM --> Client: <StreamState>Playing</StreamState>

end

end
```