**RI Playspeed=0 Recommended Approach & Estimates**

**Summary:**

Total effort to implement support for removal of Playspeed 0 is approximately one week. This includes both server and player side changes.

Testing effort is estimated at three days to verify existing tests and develop new test to verify connection stalling timeout.

**Details:**

1. Server Side Changes
   1. Remove playspeed zero from list of supported playspeeds in mpeos\_hn\_server\_profiles.c.
   2. Add mpeos\_hnServerGetConnectionStallingFlag() to mpeos\_hn.h to retrieve from platform if connection stalling is supported for specific content item and implement method in mpeos\_hn\_server\_profiles.c.
   3. Add call to this new JNI/ MPEOS method in HNStreamProtocolInfo to determine the value for http-stalling flag in DLNA Flags parameter in protocol info.
   4. Add property to hn.properties which specifies the connection inactivity timeout for idle connection which is read by HNServerSessionManager.
   5. Add connection inactivity timeout argument to HNServerSession.openSession()
   6. Replace mpe\_Bool supportConnectionStalling in mpeos\_hn.h with int32\_t connectionInactivityTimeoutSecs which is the maximum number of seconds of inactivity prior to closing the connection. If -1 is specified, it indicates no connection stalling for streaming of this content.
   7. Add logic to include connectionInactivityTimeoutSecs in HNStreamParamMediaServerHttp and associated JNI code to pass down to platform.
   8. Add new MPE\_HN\_EVT\_INACTIVITY\_TIMEOUT event to mpeos\_hn.h include this event in list of possible events in method description for mpeos\_hnPlaybackStart()
   9. Add logic in hn server.c to store new timeout parameter and pass to hn\_server\_send\_thread.c
   10. Add logic in hn\_server\_send\_thread.c to monitor the inactivity timeout and to send the MPE\_HN\_EVT\_INACTIVITY\_TIMEOUT event when inactivity reaches timeout.
   11. Add logic in RecordingStream, TSBChannelStream, and VPOPStream to handle the MPE\_HN\_EVT\_INACTIVITY\_TIMEOUT event.
2. Player Side Changes
   1. Add HNStreamProtocolInfo.isConnectionStallingSupported() which returns true if content item supports connection stalling
   2. Add logic in HNClientSession.setRate() to save rate prior to pause when called with rate = 0.0 and connection stalling is supported for content item
   3. Add two new mpeos\_hn.h methods: mpeos\_hnPlayerPlaybackPause() and mpeos\_hnPlayerPlaybackResume().
   4. Add logic in HNClientSession.setRate() to call mpeos\_hnPlayerPlaybackPause() when rate = 0.0 and connection stalling is supported for content item.
   5. Add a mpe\_MediaHoldFrameMode parameter to mpeos\_hnPlaybackStop() which is set to MPE\_MEDIA\_STOP\_MODE\_HOLD\_FRAME when platform does not support mpeos\_hnPlayerPlaybackPause().
   6. If the return code from mpeos\_hnPlayerPlaybackPause()is MPE\_HN\_ERR\_NOT\_IMPLEMENTED, call mpeos\_hnPlaybackStop()with mpe\_MediaHoldFrameMode parameter set to MPE\_MEDIA\_STOP\_MODE\_HOLD\_FRAME
   7. Add logic in HNClientSession.setRate() to check if current rate=0 and rate prior to pause = new rate, if so, call mpeos\_hnPlayerPlaybackResume().
   8. If the return code from mpeos\_hnPlayerPlaybackResume()is MPE\_HN\_ERR\_NOT\_IMPLEMENTED, call mpeos\_hnPlaybackStart()
   9. Add logic in HNClientSession.setRate() to check if current rate=0 and rate prior to pause != new rate, call mpeos\_hnPlaybackStart()
   10. Add support for new parameter in hn\_player.c mpeos\_hnPlaybackStop() to hold last frame when stopping playback if requested.
   11. Add implementation of new mpeos\_hnPlayerPlaybackPause()which will cause read thread to stop reading from socket
   12. Add implementation of new mpeos\_hnPlayerPlaybackResume()which will cause read thread to resume reading from socket
3. Additional Testing Recommended
   1. Add server tests which verifies connection stalling, disabling or enabling through mpeos\_hn\_server\_profiles.c
   2. Existing tests which exercise player side pause should be sufficient.
   3. Developer should test “hold frame” option on playback stop.

**MPEOS Changes to post to forum:**

Index: mpeos\_hn.h

===================================================================

--- mpeos\_hn.h (revision 35150)

+++ mpeos\_hn.h (working copy)

@@ -227,6 +227,18 @@

</pre>

\*\*/

MPE\_HN\_EVT\_FAILURE = MPE\_FAILURE\_UNKNOWN,

+

+ /\*\*

+ <pre>

+ \* For SERVER playback, this event indicates the platform's inability to send

+ \* the requested content for the supplied connection stalling timeout seconds.

+ \*

+ \* optionalEventData1 - N/A

+ \* optionalEventData2 - N/A

+ \* optionalEventData3 - N/A

+ </pre>

+ \*\*/

+ MPE\_HN\_EVT\_INACTIVTY\_TIMEOUT = MPE\_HN\_EVENT\_BASE + 7,

}

mpe\_HnEvent;

@@ -561,7 +573,11 @@

mpe\_Bool useServerSidePacing; ///< Requested serverside paced streaming.

mpe\_HnHttpHeaderFrameTypesInTrickMode frameTypesInTrickModes; ///< Requested frame types for trick modes.

- mpe\_Bool supportConnectionStalling;

+

+ int32\_t connectionStallingTimeoutSecs; ///< Max amount of elapsed time in seconds to keep the connection open

+ ///< without sending data; -1 if unspecified or connection stalling is

+ ///< not supported for this content

+

}

mpe\_HnStreamParamsMediaServerHttp;

@@ -909,6 +925,7 @@

\* <li>::MPE\_HN\_EVT\_PLAYBACK\_START</li>

\* <li>::MPE\_HN\_EVT\_PLAYBACK\_STOPPED</li>

\* <li>::MPE\_HN\_EVT\_FAILURE</li>

+ \* <li>::MPE\_HN\_EVT\_INACTIVTY\_TIMEOUT</li>

\* </ul>

\*

\* @return ::MPE\_HN\_ERR\_NOERR If successful. <br>

@@ -930,12 +947,14 @@

\* playback to be started that is associated with this session.

\*

\* @param playbackSession Handle to a playback session.

+ \* @param holdFrameMode the presentation mode to use once the playback has stopped

\*

\* @return ::MPE\_HN\_ERR\_NOERR If successful.<br>

\* ::MPE\_HN\_ERR\_INVALID\_PARAM If a parameter is invalid.<br>

\* ::MPE\_HN\_ERR\_OS\_FAILURE OS-specific failures.

\*\*/

-mpe\_Error mpeos\_hnPlaybackStop(mpe\_HnPlaybackSession playbackSession);

+mpe\_Error mpeos\_hnPlaybackStop(mpe\_HnPlaybackSession playbackSession,

+ mpe\_MediaHoldFrameMode holdFrameMode);

/\*\*

\* Return the MAC address associated with the network interface whose display

@@ -1389,7 +1408,38 @@

int32\_t \* framesPerSec);

///@}

+/\*\*

+ \* Returns indication if connection stalling is supported for the outgoing stream

+ \* of the requested content using supplied profile ID and mime type combination.

+ \*

+ \* @param contentLocation Indicates content location type such as

+ \* recording, TSB, etc.

+ \* @param contentDescription Structure describing content which varies

+ \* depending on content location type provided.

+ \* @param profileIDStr Retrieve supported connection stalling for

+ \* content item using this DLNA media profile.

+ \* The stack can pass "DTCP\_"-prefixed profile

+ \* ID string to indicate that the content item

+ \* will be transmitted using PCP header encapsulation.

+ \* @param mimeTypeStr Retrieve supported connection stalling for

+ \* content item using this mime type. The stack

+ \* will never pass "application/x-dtcp1" mime type here.

+ \* @param connectionSAtallingSupported Returns true if connection stalling is

+ \* supported for the outgoing content using the

+ \* using the supplied content/profile/mime type

+ \* combination.

+ \*

+ \* @return MPE\_HN\_ERR\_NOERR If successful.

+ \* @return MPE\_HN\_ERR\_INVALID\_PARAM If content/profileId/mimeType

+ \* combination is not supported.

+ \*\*/

+mpe\_Error mpeos\_hnServerGetConnectionStallingFlag(

+ mpe\_HnStreamContentLocation contentLocation, void \* contentDescription,

+ char \* profileIDStr, char \* mimeTypeStr,

+ mpe\_Bool \* connectionStallingSupported);

+///@}

+

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* \*\*\*/

/\*\*\* PLAYER FUNCTIONS \*\*\*/

@@ -1501,6 +1551,37 @@

mpe\_HnPlaybackSession playbackSession, mpe\_Bool block);

/\*\*

+ \* Pause the desired HN player playback session. Pausing a playback session is

+ \* accomplished by stopping socket reads also known as connection stalling.

+ \* The playback halts but remains the session remains open. The playback

+ \* remains in this state until it is resumed via mpeos\_hnPlayerPlaybackResume()

+ \* or server side terminates connection due to inactivity.

+ \*

+ \* @param playbackSession Handle to the media decode session to be paused

+ \*

+ \* @return MPE\_HN\_ERR\_NOERR If successful.

+ \* MPE\_HN\_ERR\_INVALID\_PARAM If a parameter is invalid.

+ \* MPE\_HN\_ERR\_OS\_FAILURE OS-specific failures.

+ \*\*/

+mpe\_Error mpeos\_hnPlayerPlaybackPause(

+ mpe\_HnPlaybackSession playbackSession);

+

+/\*\*

+ \* Resume the desired HN player playback session. Resuming a paused playback

+ \* session is accomplished by reactivating socket reads which were suspended

+ \* (referred to as connection stalling). The playback will re-initiate presentation

+ \* of video and audio as content stream is read.

+ \*

+ \* @param playbackSession Handle to the media decode session to be paused

+ \*

+ \* @return MPE\_HN\_ERR\_NOERR If successful.

+ \* MPE\_HN\_ERR\_INVALID\_PARAM If a parameter is invalid.

+ \* MPE\_HN\_ERR\_OS\_FAILURE OS-specific failures.

+ \*\*/

+mpe\_Error mpeos\_hnPlayerPlaybackResume(

+ mpe\_HnPlaybackSession playbackSession);

+

+/\*\*

\* Set the current mute state. Audio must be muted if mute is TRUE regardless

\* of content format and the gain value must be unaffected. When FALSE, audio

\* must be restored and gain must be set to the pre-mute level.

RI Playspeed=0 Analysis

**Current Implementation:**

Currently, RI Server advertises playspeed 0 in list of supported playspeeds. Playspeed 0 is handled just like any other playspeed request. HN Playspeeds mirror what is supported for local DVR playspeeds which includes playspeed zero.

**Problem - Playspeed 0 is prohibited by DLNA:**

Based on DLNA Requirement:

*Requirement [7.3.35.1]: The format of each play speed value must conform to the TransportPlaySpeed string, as specified in section 2.2.8 of* ***[****UPnP AVTransport****]****.*

The reference UPnP AV Transport requirement is:

*2.2.9 TransportPlaySpeed - …value “0” is not allowed*

In order to be compliant with DLNA, playspeed 0 must be removed. RI changes are needed on both player and server sides to support pause outside the use of playspeed 0.

**DLNA Recommended Approach:**

This is the DLNA recommendation for supporting the pause media operation:

*Requirement [7.4.58.1]: A streaming HTTP Client Endpoint should implement the Pause*

*media operation by one of the following methods:*

*• Disconnecting and Seeking: Disconnecting the HTTP connection with the intent to use*

*byte-base seek or time-based seek transport layer features as the mechanism for*

*Pause-Release.*

*• Connection Stalling: Suspending the reading of data from the HTTP connection.*

*Resuming the reading of data from the HTTP connection is the mechanism for Pause-*

*Release*

For either approach:

*Requirement [7.4.58.2]: If a streaming HTTP Client Endpoint wants to pause the current*

*media stream, it must first ensure that all of the necessary media operations and*

*information are available to resume the play (Pause-Release) of the media stream.*

*Comment: The ability to do a Pause-Release media operation depends on both the*

*Content Receiver and the Content Source sharing support for at least one of the*

*following HTTP transport features: byte-based seek, time-based seek, or Connection*

*Stalling.*

DLNA recommends that Servers should support Connection Stalling based on:

Requirement [7.4.59.3]: Comment: Using TCP flow control to stall/pause the flow of data can enable quick Pause-Release behavior. Content Sources are recommended to support the Connection Stalling method, in addition to byte-based seek or time-based seek transport layer features.

**Applicable OCAP Requirements:**

There are no direct OCAP HN requirements which dictate how playspeed zero should be handled.

The OCAP HNP Spec has the following:

Appendix B.2.2 - Trick Modes

It is recommended by [DLNA vol 1] that the connection be closed and re-opened when changing the play speed.

(can’t find the reference in DLNA).

It could be argued that the above reference implies that a player should disconnect when pausing/setting playspeed = 0 and using connection stalling would be in violation of the OCAP HNP Spec.

**Method 1 – Disconnecting and Seeking DLNA Requirements & Overview:**

Here are the applicable Disconnecting and Seeking DLNA requirements:

*Requirement [7.4.59.3]: If a streaming HTTP Client Endpoint supports the Pause media operation using the Disconnecting and Seeking method, it may perform the Pause media operation by first suspending the reading of data from the HTTP connection (as described for the Connection Stall method). When the streaming HTTP Client Endpoint detects a TCP-layer disconnect, it may perform the Pause-Release media operation using the time-based seek or byte-based seek transport layer feature that is supported by the Content Source.*

*Requirement [7.4.60.1]: If a streaming HTTP Client Endpoint performs a Pause media operation using the Connection Stalling method it must verify the http-stalling parameter in the 4th field of the res@protocolInfo is present and set to true for a content binary.*

**Method 2 - Connection Stalling DLNA Requirements & Overview:**

Here are the applicable Connection Stalling DLNA requirements:

*Requirement [7.4.60.1]: Content Sources may choose to support only the Connection Stalling method for some content binaries, such as those created by dynamic, real-time transcoding.*

*Requirement [7.4.60.3]: If the http-stalling flag is true for a content binary, then the streaming HTTP Server Endpoint must allow Connection Stalling for an indefinite amount of time on that content binary.*

*Equivalently, streaming HTTP Server Endpoints that support the Connection Stalling method for a content binary must be able to maintain the HTTP connection and must not use an HTTP connection inactivity timeout to terminate the HTTP connection.*

*Requirement [7.4.60.3]: Comment: This guideline prohibits using an HTTP-inactivity timeout to terminate HTTP connections that are being paused through Connection Stalling.*

*The guideline permits the streaming HTTP Server Endpoint to terminate HTTPconnections for the following scenarios:*

*• when the Content Receiver terminates the HTTP connection,*

*• the underlying TCP transport session is broken or disconnected,*

*• system events on the streaming HTTP Server Endpoint: user-initiated termination of streams, scheduled recording events, configurable policies for idle connections, etc.*

*Although this guideline requires a streaming HTTP Server Endpoint to allow Connection Stalling for an indefinite period of time, a streaming HTTP Server Endpoint can provide users with the ability to terminate the HTTP connections. Many details in this area are out of scope, but this guideline accounts for the following types of possibilities:*

*• A local UI, associated with the Content Source, allows the user to manually terminate the HTTP connections.*

*• The Content Source has user-configurable policies that can override the default behavior of indefinite connection stalling by terminating HTTP connections that have been inactive for a lengthy time. These guidelines do not define a minimum time but the suggested minimum HTTP inactivity timeout is 5 minutes.*

*• UPnP AV MediaServer control points invoke CMS:ConnectionComplete to terminate connections.*

*Requirement [7.3.51.1]: If the http-stalling flag is true, then the associated HTTP server must be capable of supporting the Connection Stalling method for the Pause and Pause-Release media operations on the content binary and in addition the sp-flag must be false.*

*Comment: The Connection Stalling is a mechanism where a Content Receiver and a Content Source cooperatively use standard TCP flow control to temporarily pause the transmission of data.*

*HTTP Server Endpoints are not to misinterpret HTTP-level transport inactivity as a symptom of a TCP disconnect because a properly stalled HTTP Client Endpoint will use standard TCP flow control to keep the TCP connection alive. HTTP Server Endpoints should also be careful to not overflow their local network buffers when the Connection Stalling method is being used.*

Description of TCP Connection Stalling from <http://stackoverflow.com/questions/1466307/what-is-a-tcp-window-update>:

TCP windows are used for flow control between the peers on a connection. With each ACK packet, a host will send a "window size" field. This field says how many bytes of data that host can receive before it's full. The sender is not supposed to send more than that amount of data.

The window might get full if the client isn't receiving data fast enough. In other words, the TCP buffers can fill up while the application is off doing something other than reading from it's socket. When that happens, the client would send an ACK packet with the "window full" bit set. At that point, the server is supposed to stop sending data. Any packets sent to a machine with a full window will not be acknowledged. (This will cause a badly behaved sender to retransmit. A well-behaved sender will just buffer the outgoing data. If the buffer on the sending side fills up too, then the sending app will block when it tries to write more data to the socket!) This is a TCP stall. It can happen for a lot of reasons, but ultimately it just means the sender is transmitting faster than the receiver is reading.

Once the app on the receiving end gets back around to reading from the socket, it will drain some of the buffered data, which frees up some space. The receiver will then send a "window update" packet to tell the sender how much data it can transmit. The sender starts transmitting it's buffered data and traffic should flow normally.

**General Implementation Effort required regardless of pause method used:**

**Server Changes:**

* Remove playspeed 0 from list - mpeos\_hn\_server\_profiles.c

**Player Changes:**

* Logic remains same up to the point where the HTTP request generation begins which is in HNClientSession.java
* Most changes will be within HNClientSession.java
* Need to maintain previous time and byte positions so it is available when rate = 0 is requested
* Even though there is no platform support needed for rate = 0, can’t release pipeline since video device is still needed

**Method 1 – *Disconnecting and Seeking* Required Implementation Effort**

There are no additional changes required to support this pause method on either the Server or Player side.

**Method 2 - *Connection Stalling* Required Implementation Effort**

**Server Changes:**

* Socket options need to be modified with appropriate timeout
* Inactivity timeout within the stack code will need to be adjusted
* Additional config parameter to specify timeout when connection stalling for idle connections
* Update protocol info to indicate support for connection stalling

**Player Changes:**

* Need to ensure the socket is not closed when rate change of zero is requested
* Need socket timeout option adjusted if connection stalling is desired
* Verify requested content supports connection stalling

**Application Changes & Required Test Support:**

* No changes required to RiExerciser
* Standard integration tests should cover pause
* Additional test development will be required if Connection Stalling method is used

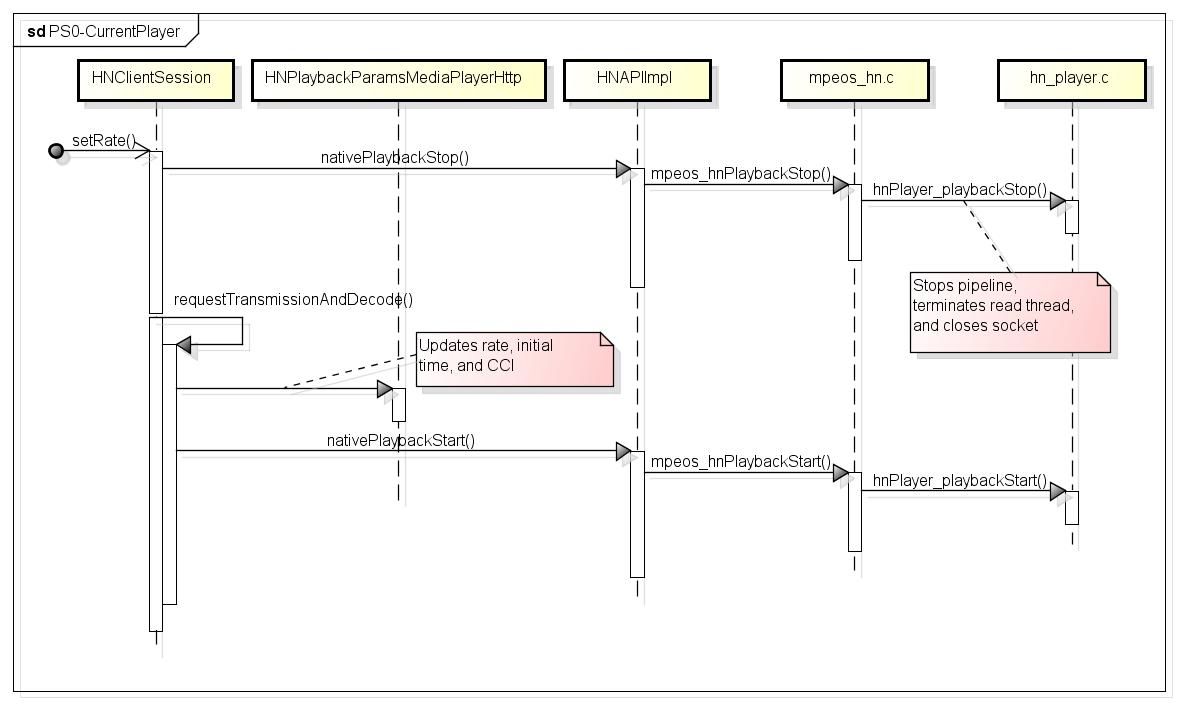
****

Figure - Sequence Diagram of Current Implementation

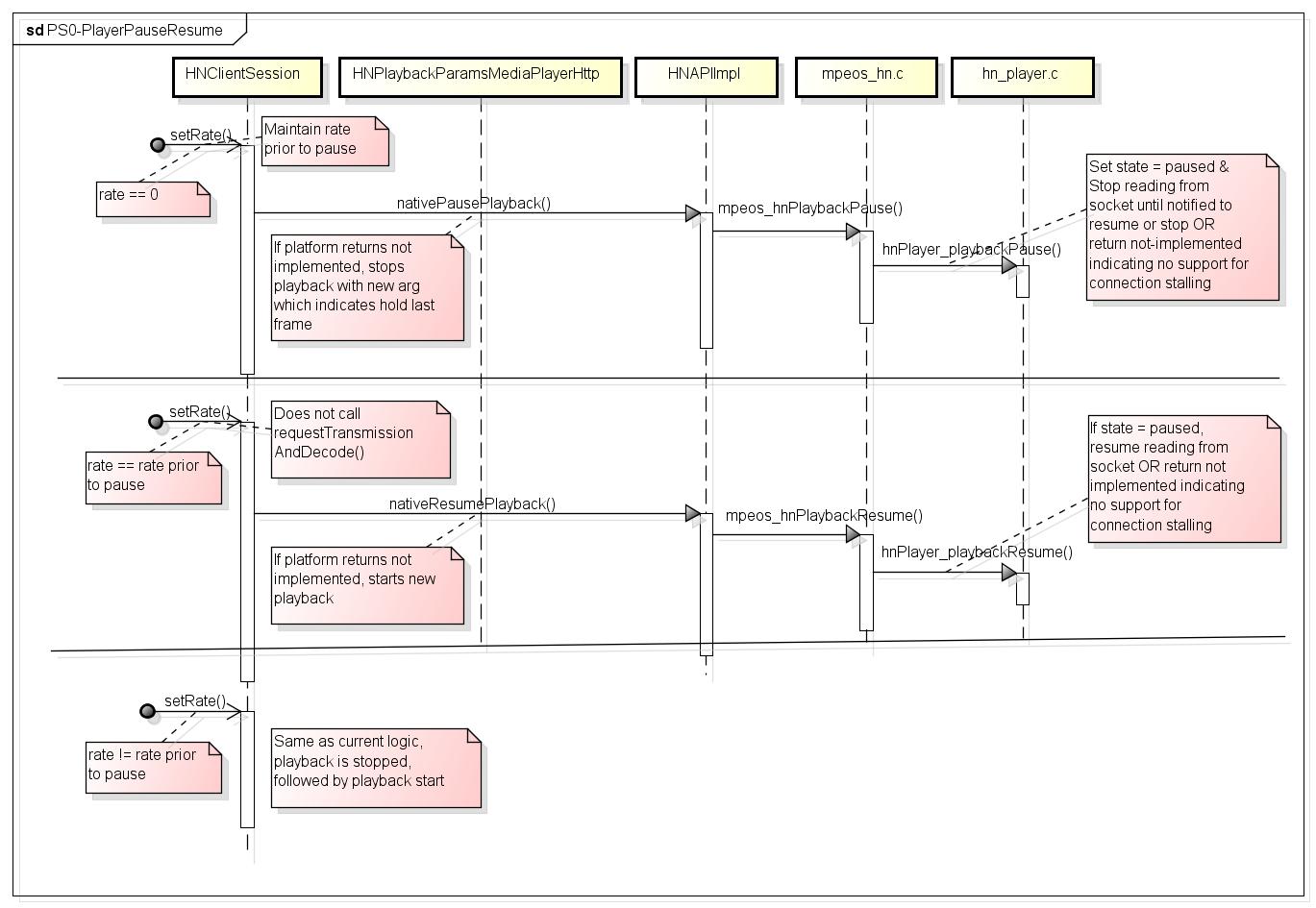
****

Figure - Sequence Diagram of using new Pause/Resume Methods

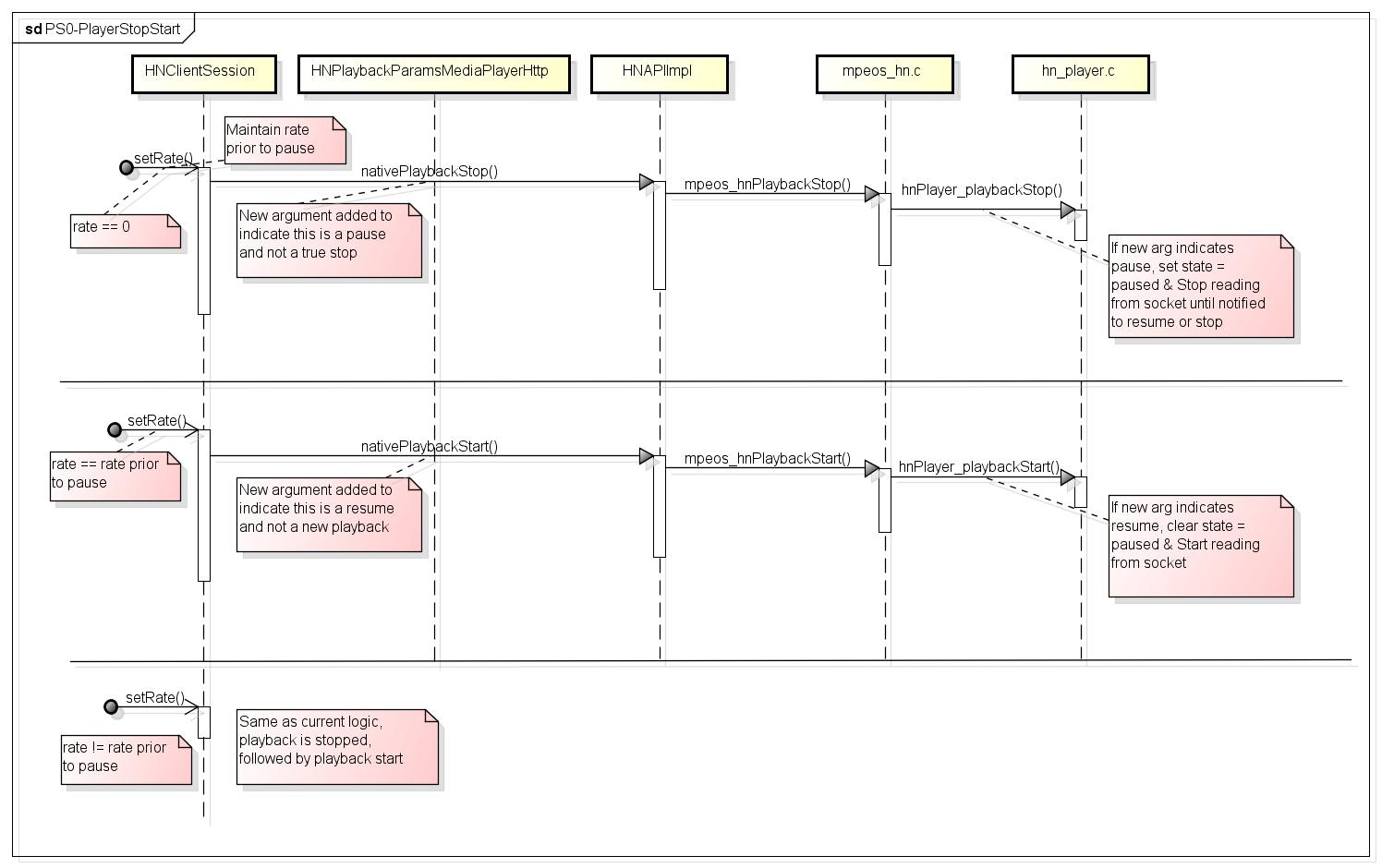
****

Figure - Sequence Diagram using Playback Stop & Start