**3GPP TSG-SA4 Meeting #108e *S4-200553***

**e-meeting, 2 - 9 April 2020**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *CR-Form-v12.0* | | | | | | | | |
| **Pseudo CHANGE REQUEST** | | | | | | | | |
|  | | | | | | | | |
|  | **26.512** | **CR** | **<CR#>** | **rev** | **-** | **Current version:** | **1.0.1** |  |
|  | | | | | | | | |
| *For* [***HE******LP***](http://www.3gpp.org/3G_Specs/CRs.htm#_blank)*on using this form: comprehensive instructions can be found at* [*http://www.3gpp.org/Change-Requests*](http://www.3gpp.org/Change-Requests)*.* | | | | | | | | |
|  | | | | | | | | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Proposed change affects:*** | UICC apps |  | ME | **X** | Radio Access Network |  | Core Network | **X** |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | |
| ***Title:*** | Client APIs for 5GMS | | | | | | | | | |
|  |  | | | | | | | | | |
| ***Source to WG:*** | Qualcomm Incorporated | | | | | | | | | |
| ***Source to TSG:*** | SA4 | | | | | | | | | |
|  |  | | | | | | | | | |
| ***Work item code:*** | 5GMS3 | | | | |  | ***Date:*** | | | 2020-03-31 |
|  |  | | | |  | |  | | |  |
| ***Category:*** | **B** |  | | | | | ***Release:*** | | | Rel-16 |
|  | *Use one of the following categories:* ***F*** *(correction)* ***A*** *(mirror corresponding to a change in an earlier release)* ***B*** *(addition of feature),* ***C*** *(functional modification of feature)* ***D*** *(editorial modification)*  Detailed explanations of the above categories can be found in 3GPP [TR 21.900](http://www.3gpp.org/ftp/Specs/html-info/21900.htm). | | | | | | | | *Use one of the following releases: Rel-8 (Release 8) Rel-9 (Release 9) Rel-10 (Release 10) Rel-11 (Release 11) Rel-12 (Release 12)* *Rel-13 (Release 13) Rel-14 (Release 14) Rel-15 (Release 15) Rel-16 (Release 16)* | |
|  |  | | | | | | | | | |
| ***Reason for change:*** | | Client APIs not existing | | | | | | | | |
|  | |  | | | | | | | | |
| ***Summary of change:*** | | Adds Client APIs | | | | | | | | |
|  | |  | | | | | | | | |
| ***Consequences if not approved:*** | | Fixes not done | | | | | | | | |
|  | |  | | | | | | | | |
| ***Clauses affected:*** | |  | | | | | | | | |
|  | |  | | | | | | | | |
|  | | **Y** | **N** |  | | | |  | | |
| ***Other specs*** | |  | **X** | Other core specifications | | | | TS/TR ... CR ... | | |
| ***affected:*** | |  | **X** | Test specifications | | | | TS/TR ... CR ... | | |
| ***(show related CRs)*** | |  | **X** | O&M Specifications | | | | TS/TR ... CR ... | | |
|  | |  | | | | | | | | |
| ***Other comments:*** | | This pCR assumes that document S4-200509 is agreed. | | | | | | | | |
|  | |  | | | | | | | | |
| ***This CR's revision history:*** | |  | | | | | | | | |

**===== CHANGE =====**

## 4.7 Procedures of the M6d (UE Media Session Handling) interface

The following states are defined for a media session handler:

* IDLE In this state the Media Session handler does not have any registered 5G-Aware application that is served. It does also not keep track service definition.
* REGISTERED In this state the Media Session handler has registered the 5G-Aware Application, it communicates with the Media AF and keeps the service information up to date.
* ACTIVE In this state the Media Session Handler provides all services to of the REGISTERED state and also provides the streaming service to the application. In this state the media session handler sends callback notifications to the application and the Media Player.
* NON\_AVAILABLE In this state the media Session handler is not available and an 5GMSd-aware application cannot register with the MBMS client.
* STALLED In this state the Media session handler client provides all services of the REGISTERED state, but the streaming service on M4d is at least temporarily stalled.

At the minimum following information from the User Service Description triggers is maintained

Table X: Parameters of Media Session Hander for DASH Streaming Service

| Internal Parameters | | | Definition |
| --- | --- | --- | --- |
| \_app[] | | | The media session handler maintains a parameter set per registered app |
|  | \_appId | | A unique ID provided by the application and assigned to the app. |
|  | \_service[] | | The media session handler maintains a parameter list per service. In this context the list is assigned also to one app, but an implementation may share the internal parameter list assigned to a service across multiple apps. |
|  |  | \_serviceID | The service ID for a Streaming Application service for which the media session handling is applied. |
|  |  | \_serviceName[]  \_name  \_lang | The service name, possibly expressed in different languages. |
|  |  | \_MPD  \_IS[]  \_mpdURI | The latest MPD associated to the service  The Initialization Segments for the Media Presentation  The URI which is provided to the application for initiating the DASH Media Presentation. |
|  |  | \_sessionSchedule[]  \_start  \_stop | Documents the session schedule for this session. Only sessionSchedule records should be included for which the value of the \_stop time is in the future. |

Table Y provides an overview over the methods defined for the Streaming Delivery Application Service API. Different types are differentiated, namely state changes triggered by the Application, status query of the application to the media session handler, parameter updates as well as notifications from the media session handler. The direction of the main communication flow between Application (A) and Media Session Handler Client (h) is provided.

Note that the information may be provided from the application or the media player.

Table X: Methods defined for Streaming Delivery Application Service API

| Method | Type | Direction | Brief Description |
| --- | --- | --- | --- |
| registerStreamingApp | State change | A -> H | Application/Player registers a callback listener with the media session handlert |
| deregisterStreamingApp | State change | A -> H | Application/Player deregisters with the media session handler |
| startStreamingService | State change | A -> H | Starts streaming service |
| stopStreamingService | State change | A -> H | Stop streaming service |
| registerStreamingResponse | Update to parameter list | H> A | The response to the Application/Player streaming service register API |
| serviceStarted | Notification | H -> A | Notification to Application when the media session handling has started. |
| streamingServiceError | Notification | H -> A | Notification to Application/Player when there is an error with the access of service |
| serviceStalled | Notification | CH-> A | Notification to Application/Player that access resources on M4d failed. |

Beyond the above states, procedures and notifications on M6d, additional ones may be defined that result from any of the scenarios that are documented.

**===== CHANGE =====**

## 4.8 Procedures of the M7d (UE Media Player) interface

### 3.2 Proposal for 5G Media Streaming

The following states are defined for a media session handler:

* IDLE In this state the Media Session handler does not have any registered 5G-Aware application that is served. It does also not keep track service definition.
* REGISTERED In this state the Media Session handler has registered the 5G-Aware Application, it communicates with the Media AF and keeps the service information up to date.
* ACTIVE In this state the Media Session Handler provides all services to of the REGISTERED state and also provides the streaming service to the application. In this state the media session handler sends callback notifications to the application and the Media Player.
* NON\_AVAILABLE In this state the media Session handler is not available and an 5GMSd-aware application cannot register with the MBMS client.
* STALLED In this state the Media session handler client provides all services of the REGISTERED state, but the streaming service on M4d is at least temporarily stalled.

At the minimum following information from the User Service Description triggers is maintained

Table X: Parameters of Media Session Hander for DASH Streaming Service

| Internal Parameters | | | Definition |
| --- | --- | --- | --- |
| \_app[] | | | The media session handler maintains a parameter set per registered app |
|  | \_appId | | A unique ID provided by the application and assigned to the app. |
|  | \_service[] | | The media session handler maintains a parameter list per service. In this context the list is assigned also to one app, but an implementation may share the internal parameter list assigned to a service across multiple apps. |
|  |  | \_serviceID | The service ID for a Streaming Application service for which the media session handling is applied. |
|  |  | \_serviceName[]  \_name  \_lang | The service name, possibly expressed in different languages. |
|  |  | \_MPD  \_IS[]  \_mpdURI | The latest MPD associated to the service  The Initialization Segments for the Media Presentation  The URI which is provided to the application for initiating the DASH Media Presentation. |
|  |  | \_sessionSchedule[]  \_start  \_stop | Documents the session schedule for this session. Only sessionSchedule records should be included for which the value of the \_stop time is in the future. |

Table Y provides an overview over the methods defined for the Streaming Delivery Application Service API. Different types are differentiated, namely state changes triggered by the Application, status query of the application to the media session handler, parameter updates as well as notifications from the media session handler. The direction of the main communication flow between Application (A) and Media Session Handler Client (h) is provided.

Note that the information may be provided from the application or the media player.

Table X: Methods defined for Streaming Delivery Application Service API

| Method | Type | Direction | Brief Description |
| --- | --- | --- | --- |
| registerStreamingApp | State change | A -> H | Application/Player registers a callback listener with the media session handlert |
| deregisterStreamingApp | State change | A -> H | Application/Player deregisters with the media session handler |
| startStreamingService | State change | A -> H | Starts streaming service |
| stopStreamingService | State change | A -> H | Stop streaming service |
| registerStreamingResponse | Update to parameter list | H> A | The response to the Application/Player streaming service register API |
| serviceStarted | Notification | H -> A | Notification to Application when the media session handling has started. |
| streamingServiceError | Notification | H -> A | Notification to Application/Player when there is an error with the access of service |
| serviceStalled | Notification | CH-> A | Notification to Application/Player that access resources on M4d failed. |

Beyond the above states, procedures and notifications on M6d, additional ones may be defined that result from any of the scenarios that are documented.

**===== CHANGE =====**

## 5.X DASH Player APIs

Based on the cyan procedures summarized above 5GMSd aware application predominantly uses M7d to communicate with the 5GMSd Client. Using the dash.js APIs as introduced in S4-191128, relevant APIs are:

#### **[attachSource(urlOrManifest)](http://cdn.dashjs.org/latest/jsdoc/module-MediaPlayer.html" \l "attachSource)**

Use this method to set a source URL to a valid MPD manifest file OR a previously downloaded and parsed manifest object. Optionally, can also provide protection information

#### **[pause()](http://cdn.dashjs.org/latest/jsdoc/module-MediaPlayer.html" \l "pause)**

This method will call pause on the native Video Element.

#### **[play()](http://cdn.dashjs.org/latest/jsdoc/module-MediaPlayer.html" \l "play)**

The play method initiates playback of the media defined by the [attachSource()](http://cdn.dashjs.org/latest/jsdoc/module-MediaPlayer.html" \l "attachSource) method. This method will call play on the native Video Element.

#### **[preload()](http://cdn.dashjs.org/latest/jsdoc/module-MediaPlayer.html" \l "preload)**

Causes the player to begin streaming the media as set by the [attachSource()](http://cdn.dashjs.org/latest/jsdoc/module-MediaPlayer.html" \l "attachSource) method in preparation for playing. It specifically does not require a view to be attached with [attachView()](http://cdn.dashjs.org/latest/jsdoc/module-MediaPlayer.html" \l "attachSource) to begin preloading. When a view is attached after preloading, the buffered data is transferred to the attached mediaSource buffers.

#### **[reset()](http://cdn.dashjs.org/latest/jsdoc/module-MediaPlayer.html" \l "reset)**

Sets the MPD source and the video element to null. You can also reset the MediaPlayer by calling attachSource with a new source file.

Calling this method is all that is necessary to destroy a MediaPlayer instance.

#### **[seek(value)](http://cdn.dashjs.org/latest/jsdoc/module-MediaPlayer.html" \l "seek)**

Sets the currentTime property of the attached video element. If it is a live stream with a timeShiftBufferLength, then the DVR window offset will be automatically calculated.

Other procedures may be used, but those seem to be the most relevant ones.

In particular, the Application should allow the Media Session handler to actively use a set of get methods. Examples include

#### **[getAverageThroughput(type)](http://cdn.dashjs.org/latest/jsdoc/module-MediaPlayer.html" \l "getAverageThroughput)**

Returns the average throughput computed in the ABR logic

#### **[getBufferLength(type)](http://cdn.dashjs.org/latest/jsdoc/module-MediaPlayer.html" \l "getBufferLength)**

The length of the buffer for a given media type, in seconds. Valid media types are "video", "audio" and "fragmentedText". If no type is passed in, then the minimum of video, audio and fragmentedText buffer length is returned. NaN is returned if an invalid type is requested, the presentation does not contain that type, or if no arguments are passed and the presentation does not include any adaption sets of valid media type.

#### **[getCurrentLiveLatency()](http://cdn.dashjs.org/latest/jsdoc/module-MediaPlayer.html" \l "getCurrentLiveLatency)**

Current live stream latency in seconds. It is the difference between current time and time position at the playback head.

#### **[getDashMetrics()](http://cdn.dashjs.org/latest/jsdoc/module-MediaPlayer.html" \l "getDashMetrics)**

Returns the DashMetrics.js Module. You use this Module to get access to all the public metrics stored in dash.js

#### **[getPlaybackRate()](http://cdn.dashjs.org/latest/jsdoc/module-MediaPlayer.html" \l "getPlaybackRate)**

Returns the current playback rate.

Other API calls may be defined to be exposed to the media session handler.

One the Media Player is active, the Media Player is expected provide notifications. These notifications may be provided to the application but can also be redirected to the activated Media Session Handler. Relevant Notifications for the Application are:

#### **MANIFEST\_LOADED**

Triggered when the manifest load is complete

#### **PLAYBACK\_ENDED**

Sent when playback completes.

#### **PLAYBACK\_PAUSED**

Sent when playback is paused.

#### **PLAYBACK\_PLAYING**

Sent when the media begins to play (either for the first time, after having been paused, or after ending and then restarting).

Other set of notifications may be of relevant for the application.

A set of notifications that may be delegated to the media session handler.

#### **BUFFER\_EMPTY**

Triggered when the video element's buffer state changes to stalled. Check mediaType in payload to determine type (Video, Audio, FragmentedText).

#### **BUFFER\_LEVEL\_STATE\_CHANGED**

Triggered when the video element's buffer state changes, either stalled or loaded. Check payload for state.

#### **METRIC\_ADDED**

Triggered every time a new metric is added.

#### **METRIC\_CHANGED**

Triggered when an individual metric is added, updated or cleared.

#### **METRIC\_UPDATED**

Triggered every time a metric is update

#### **METRICS\_CHANGED**

Triggered anytime there is a change to the overall metrics.

#### **PLAYBACK\_ENDED**

Sent when playback completes.

#### **PLAYBACK\_PAUSED**

Sent when playback is paused.

#### **PLAYBACK\_PLAYING**

Sent when the media begins to play (either for the first time, after having been paused, or after ending and then restarting).

#### **PLAYBACK\_RATE\_CHANGED**

Sent when the playback speed changes.

#### **PLAYBACK\_STALLED**

Sent when the video element reports stalled

#### **QUALITY\_CHANGE\_RENDERED**

Triggered when the new ABR quality is being rendered on-screen.

#### **QUALITY\_CHANGE\_REQUESTED**

Triggered when an ABR up /down switch is initiated; either by user in manual mode or auto mode via ABR rules.

Other notifications may be of relevance as well should be collected.