7 Synchronization with timestamps

For data sinks with passive rendering, PVPlayer engine must output the media data properly in time. This is accomplished by evaluating the timestamp associated with each media data with the current value of the playback clock. If the media data's timestamp is equal to the current clock time, the media data is in synchronization and rendered. If the timestamp is less than the clock time, the media data is early. If the media data is greater than the current clock time, the media data is late. What happens to media data that is early or late depends on how PVPlayer engine is configured. In most uses, early media data is held until it becomes in synchronization and late media data is dropped without being rendered. But in some configurations, the late media data might be rendered as well.

Another area in timestamp synchronization that could be configured is the margin for being in synchronization. In ideal situation, the margin is 0 where timestamp must be equal to the clock time to be in synchronization. But due to various factors such as clock resolution and active object scheduling resolution, the margin must be larger or the media data being in synchronization would be missed. The synchronization point could also be offset to deal with some fixed latency in the rendering. For example, if a video render device requires certain time to actually display the video frame, the synchronization might happen at an earlier time so when data is sent to the device and the actual display would occur when the timestamp value equaled the clock time.

This synchronization functionality is performed in the data sinks with such support. These data sinks take a reference to the playback clock from PVPlayer engine and reads the media data timestamp from each PVMF media data object. PvPlayer engine determines if a data sink has synchronization support in the capability exchange process.

7.1 Synchronization with flow controlling data sink

If the data sink has flow control and the media data for that sink is rendered continuously, PVPlayer engine needs to take the data output rate of the data sink into account. If there is a temporal difference between how the flow controlled media data is actually rendered by the data sink and PVPlayer engine's playback clock, AV synchronization mismatch might appear between the media tracks. Depending on the severity of the mismatch, the problem might be detectable by the person viewing the multimedia playback.

To prevent such a temporal difference, the flow controlled data sink would perform adjustments to the playback clock based on information of media data rendering. Such information could be feedback from the rendering device on how much of data has been actually rendered or the time of last rendered media data. The data sink is responsible for converting the correction information to a format that acceptable to the clock adjustment method of OSCL clock. An example of a data sink with flow controlling rendering is a PCM audio output sink node. PCM audio data is continuous and audio devices typically output the PCM audio data by some fixed sampling rate. The clock controlling the output could be different from the clock source that PVPlayer engine's playback clock uses so there could a difference in how time progresses between the two. Over time, the difference could accumulate and other media data (e.g. video, text) could be rendered out-of-synchronization with the audio. Audio devices could return the number of PCM samples rendered or the system time when the last audio media data was rendered and the audio output data sink node could use this information to adjust PVPlayer engine's playback clock.

7.2 Synchronization with combination

In typical multimedia playback scenarios, PVPlayer SDK will interact with data sinks of both type: one which relies on timestamp only and one which relies on flow control. In such cases, the data sink with flow control is allowed to adjust the playback clock so all media data is kept in synchronization with each other.

7.3 Faster or slower than "real-time"

Since all media output rate in PVPlayer SDK is controlled by PVPlayer engine's playback clock, the playback rate can be changed by modifying the pacing of the playback clock. The playback clock is based on OSCL clock class so it uses a timebase to know how much time has elapsed. Typically the timebase uses the system tickcount or some other system timing function to report how much time has elapsed in microseconds. By using such a timebase, PVPlayer SDK will playback the media data in "real-time". But if the timebase was modified to report elapsed time as being faster than or slower than "real-time" then playback could occur faster or slower respectively. By making such modifications to the timebase, PVPlayer SDK provides such features as fast forward (faster than "real-time"), slow motion (slower than "real-time"), or frame-by-frame (slower than "real-time" without set rate).

When playback rate is modified as such, media tracks with data sinks that can only work in one fixed rate must be disabled since those sinks cannot play the data faster or slower. Typical data sink with such a limitation is the audio output device data sink. The audio output device usually can only accept audio data in a fixed sampling rate. If the playback rate changed, the audio data would be fed to the device too fast or too slow and could cause the data sink to overflow or underflow with undesirable effects. Therefore in such a case, PVPlayer engine will disable media tracks with data sinks with such restrictions. PvPlayer engine will determine if the data sink can handle different playback rates by querying its capabilities.

8 Playback Control

PVPlayer SDK provides methods to control the multimedia playback. This section describes what occurs inside PVPlayer engine when these control commands are issued.

8.1 Starting and Stopping

When starting playback, PVPlayer engine commands the PVMF nodes in the data-flow graph to start and then starts the playback clock. When stopping, PVPlayer engine stops the playback clock and commands the PVMF nodes in the data-flow graph to stop. Doing so flushes all media data in the data-flow graph.

8.2 Pausing and resuming

When paused, the playback clock is not progressing forward and media data is not rendered via the data sinks. But unlike being stopped, media data is still queued in the data-flow graph ready to be restarted. Resuming takes PVPlayer SDK out of paused state to have playback clock moving forward and media data to be rendered again.

When pausing, PVPlayer engine pauses the playback clock and commands the PVMF nodes in the data-flow graph to pause.
When resuming, PVPlayer engine restarts the PVMF nodes in the data-flow graph and then restarts the playback clock.

8.3 Repositioning

Repositioning is the changing of playback position in the clip during playback. An example would be to be playing the clip at 10 seconds and then immediately jumping to the clip at 30 seconds and continuing playback. PvPlayer SDK handles repositioning as a change in the data source's media data position and continuing playback. Since the playback clock does not jump during playback, the data source or PVMF node responsible for providing the timestamp for the media stamp adjusts the media data timestamp to maintain this requirement.
For example, playback has been started and currently the playback position is at 30 seconds. If the playback is repositioned to the clip's time (normal playback time, NPT) of 15 seconds, the playback clock is still kept at 30 seconds and media data will be sent from clip at 15 seconds but the timestamp will continue to be from 30 seconds. After 30 more seconds, the NPT is at 45 seconds, but the playback clock and media data timestamp would be at 60 seconds. At this point, if a forward repositioning to clip's time of 90 second occurs, media data will be sent from clip at 90 seconds but the playback clock and media data timestamp will still be at 60 seconds. The table below lists this sequence.

Event	Playback clock	Clip time (NPT)
Start playback	0	0
Playback for 30 sec	30	30
Reposition to 15 sec in clip	30	15
Playback for 30 sec	60	45
Reposition to 90 sec in clip	60	90
Playback for 30 sec	90	120

The example above represents an ideal repositioning scenario. When repositioning with some data sources, PVPlayer SDK would not be able to directly reposition to specified position due to media data limitations or data source restrictions. Example of such limitation is the time resolution of the media data (e.g. audio frame) and limited seek positions (e.g. I-frames in video). If PVPlayer SDK is handling such data, there could be a transition period in reposition where additional media data might be generated and processed. PVPlayer SDK would behave to minimize such transition period but some artifacts of the transition might be unavoidable.

The PVPlayer SDK could handle the reposition transition in one of several ways and PVPlayer SDK provides methods to configure this. One such configuration deals with repositioning for video with limited seek positions (e.g. M4v). In such video data, one cannot go and playback from any frame since frames are dependent on the previously decoded frame. Playback has to start from certain frames which are not dependent on previous frames. So when repositioning, if jump-to location is at one of these frames that are not dependent on the previous frame, then playback can continue from that frame. If not, then PVPlayer SDK must go to one of these non-dependent frames before the requested repositioning position and decode the frames in between before continuing playback. For best quality, PVPlayer SDK should always go to one of these independent video frames. But if availability of these independent frames are limited, PVPlayer SDK might take some time to decode the in-between frames. In such case, the better user experience might be to just decode from a dependent frame at the requested repositioning point while sacrificing video quality. To allow the PVPlayer SDK behavior for the transition to be configurable by the user, PVPlayer SDK provides a way to configure whether to always go to the independent frame or not and the size of window to look for the independent frames via the capability-and-configuration interface (refer to the next section). If not always going to the independent frame, playback will start from a dependent frame unless there is an independent frame at the requested repositioning position. If always going to independent frame and the window is non-existent, then PVPlayer SDK will always look for the independent frame that is before the requested repositioning position. Between those two extremes, if the independent frame falls in the specified window then repositioned playback will start there. If such a frame is not found in the window, the first dependent frame past edge of the window would be used as the starting point for the repositioning. The diagrams below illustrate how the windowing works. In the first diagram, the independent frame (sync point) is outside of the window so PVPlayer engine will reposition to the edge of the window (new position).

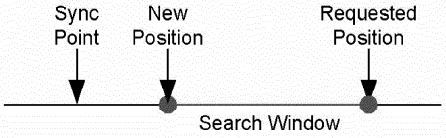

Figure 7: Independent Frame is Outside of Window

In the second diagram, the independent frame (sync point) is inside of the window so PVPlayer engine will reposition to the same position (new position)

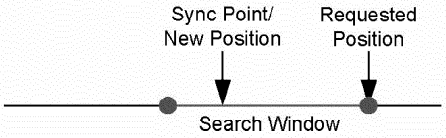

Figure 8: Independent Frame is Inside Window

The flow chart below describes how repositioning would be performed by PVPlayer engine based on the reposition configuration.

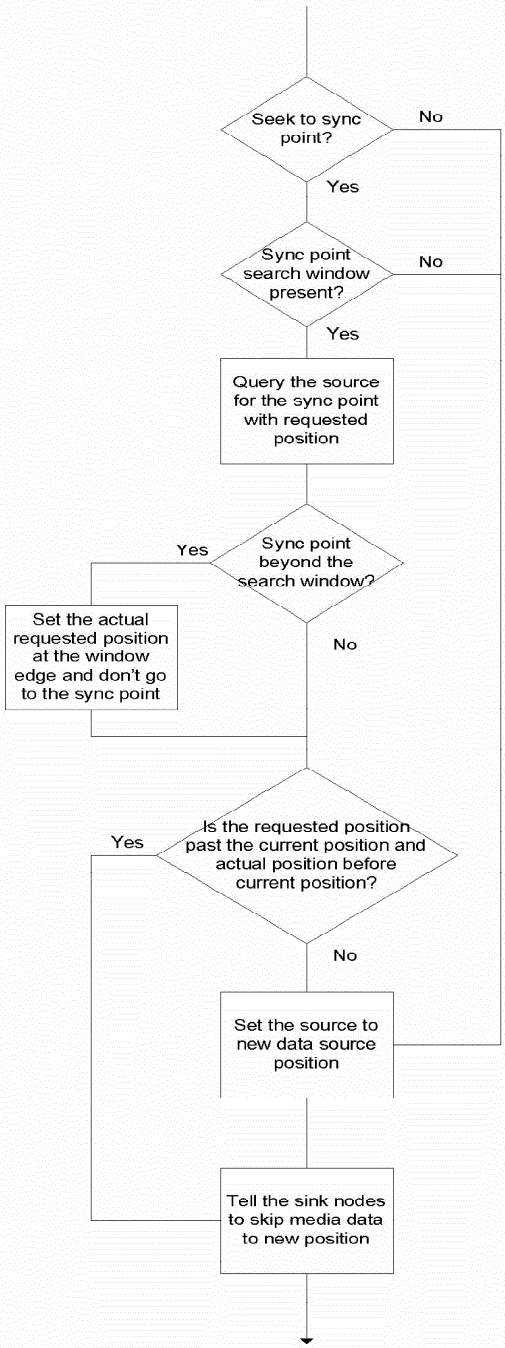

Figure 9: Reposition Processing Flow Chart

The reposition flow chart also incorporates a forward repositioning optimization when always going to an independent frame and the search window exists. If the requested repositioning position is past the current playback position and the position of the independent frame is before the current playback position, PVPlayer engine will not modify the source data position. This optimization saves unnecessary data transfer and processing between the independent frame position and the current playback position when repositioning.

Another configurable repositioning behavior but is not show in the flowchart is the option to always start rendering the media data at the requested reposition position. If this behavior is enabled and PVPlayer engine will always tell the sink nodes to skip media data up to the requested reposition position irrespective of where the source node starts sending the media data from.

Another deviation from the ideal repositioning scenario is the repositioning occurring when there are media data in the data-flow graph that are waiting to be rendered by the data sinks. This occurs if the nodes in the data-flow graph process media data ahead of the playback rate to have the media data for rendering in time and the repositioning command is not known beforehand. When repositioning, these media datas become obsolete and should not be rendered. To prevent these obsolete media data from being rendered, PVPlayer engine queries the data source node for the starting timestamp for media data after repositioning. The data source node knows this information since the data source node controls the media data going into the data-flow graph. The data source node calculates the repositioning timestamp as the next time value past the last media data sent into the data-flow graph. Then the PVPlayer engine stops and sets the playback clock to this starting timestamp and commands all data sink nodes with synchronization support to flush media data before the new repositioning timestamp. The playback clock is started again when all data sink nodes report old media data has been flushed. With this feature, the repositioning example for the ideal case would be changed to the following:

Event	Playback clock	Clip time (NPT)
Start playback	0	0
Playback for 30 sec	30	30
Reposition to 15 sec in clip but 2 sec worth of data in data-flow graph	32	15
Playback for 30 sec	62	45
Reposition to 90 sec in clip but 4 sec worth of data in data-flow graph	66	90
Playback for 30 sec	96	120

Repositioning use cases can be divided into three categories. First one is offset playback where the starting position is known before playback starts. Second one is “edit-list” playback in which when and where to reposition are known beforehand. The last use case is random positioning playback in which where to reposition is known but when to reposition is not known until the command is requested.

In the OSCL-based interface to PVPlayer engine, all three repositioning types can be realized by the SetPlaybackRange() API. Depending on the PVPlayer engine state when SetPlaybackRange() command is issued and the parameters passed in, the PVPlayer SDK user can perform all three repositioning types in playback.
SetPlaybackRange() API has two parameters for the beginning playback position and ending playback position. When the command is accepted, PVPlayer engine will play the media data between these two positions. SetPlaybackRange() also has a flag to specify whether to activate the new range immediately or queue for activation later when the current playback range completes. PvPlayer engine can only queue

one playback range at any given time so if multiple playback ranges are queued at once, only the last one queued will be actually activated. For offset playback and random positioning use cases, the flag is set to immediate activation. For “edit-list” playback, the flag is set for queuing.

Begin and end playback position parameters are allowed to be indeterminant. In such case, the beginning of the clip (time 0) and end of clip (clip duration) will replace begin and end positions, respectively. The only exception is when SetPlaybackRange() is called during playback with begin position being indeterminate and flag set for immediate activation. In such scenario, the end position will be modified without interrupting the playback (i.e. will not random position to beginning of clip).

SetPlaybackRange() can be called in most engine states, however, for performance reasons it should be called as early as possible. For example if playback is to start from 30 seconds instead of 0 seconds it would be possible to reposition after Init, after Prepare or after Start has been issued. The later two cases are inefficient because data has already been retrieved and processed and now also needs to be flushed. In this case it would be best to call SetPlaybackRange() before calling Prepare() to avoid unnecessary calculations and improve start up speed.

9 Capability Query and Configuring Settings

PVPlayer engine utilizes the PVMF capability-and-configuration interface to allow the application to access and modify engine and node settings not exposed by the player interface. The extension interface (PvmiCapabilityAndConfig) is exposed via the player API, QueryInterface(), by requesting with the UUID associated with the interface. Using the returned interface pointer, the application can query, verify, and set settings at the engine and node levels. At the node level, the node being used by the engine must support the capability-and-configuration interface as well for node settings to be accessible to the application.

Capability-and-configuration interface uses key strings in PacketVideo Extended MIME String (PvXms) format to specify the settings of interest. PvXms extends the standard MIME string format by allowing additional levels of subtype strings all separated by the slash character. Using key strings adds complexity in parsing but allows flexibility and extensibility for settings without greatly modifying code when settings are added, removed, or modified. In addition to specifying the setting of interest, the key string also provides information on value returned with the string in a key-value pair (KVP). The "type" parameter in the key string tells the user of the KVP whether there is a valid value if "type=value". The "valtype" parameter in the key string tells the user of the KVP what the value type is so the appropriate union member can be accessed in the KVP.

9.1 PVPlayer Engine Key Strings

All key strings at the PVPlayer engine level start with "x-pvmf/player". The following key strings are currently supported in the player engine

Key Strings With Value Type	Description
x-pvmf/player/pbpos_units;valtype=char*	Playback position units specified with strings ("PVPPBPOSUNIT_MILLISEC", "PVPPBPOSUNIT_SEC", "PVPPBPOSUNIT_MIN", "PVPPBPOSUNIT_FILEOFFSET")
x-pvmf/player/pbpos_interval;valtype=uint32	The interval between playback position info event. Integer value in milliseconds.
x-pvmf/player/endtimecheck_interval;valtype=uint32	The interval between the end time check routine. Integer value in milliseconds.
x-pvmf/player/seektosyncpoint;valtype=bool	The flag to specify whether to always seek to the closest sync point when repositioning.
x-pvmf/player/skiptorequestedpos;valtype=bool	The flag to specify whether to always start playback from the requested begin position (i.e. skip frames when sync point doesn't match the requested position)
x-pvmf/player/renderskipped;valtype=bool	The flag to specify whether to render the skipped frames
x-pvmf/player/syncpointseekwindow;valtype=uint32	If seeking to closest sync point, this parameter specifies how far to search back in milliseconds. If the sync point is not present in the specified window, playback would continue from the window boundary. Value of 0 means no window.

x-pvmf/player/nodecmd_timeout;valtype=uint32	The timeout limit for all node commands in milliseconds.
x-pvmf/player/nodedataqueuing_timeout;valtype=uint32	The timeout limit for engine to wait for data sink nodes to report data ready event in milliseconds
x-pvmf/player/productinfo/productname;valtype=char*	The product name string. String would be set to some default during build time but should be modifiable
x-pvmf/player/productinfo/partnumber;valtype=char*	The part number string for the product. String would be set to some default during build time but should be modifiable
x-pvmf/player/productinfo/hardwareplatform;valtype=char*	The hardware platform string for the product. Could be sent to servers. String would be set to some default during build time but should be modifiable
x-pvmf/player/productinfo/softwareplatform;valtype=char*	The general platform string for the product. String would be set to some default during build time but should be modifiable
x-pvmf/player/productinfo/device;valtype=char*	The device info string. Could be sent to servers. String would be set to some default during build time but should be modifiable

9.2 Node Level Key Strings

The node level key strings available during PVPlayer engine usage depends on PVMF nodes being used by the PVPlayer engine at that time and the key strings supported by a particular node. For node level key strings, PVPlayer engine acts as a router to pass any requests to the appropriate node. Currently, PVPlayer engine performs a hardcoded mapping from key sub-string to certain nodes, but in the future, PVPlayer engine and nodes will determine the mapping at runtime using a registration scheme.

Currently, the key string mapping to nodes is as follows in PVPlayer engine.

Key Sub-String	**Node Type**
x-pvmf/video/decoder	Video decoder node then video sink node
x-pvmf/audio/decoder	Audio decoder node than audio sink node
x-pvmf/video/render	Video sink node
x-pvmf/audio/render	Audio sink node
x-pvmf/net	Data source node (typically streaming / download)
x-pvmf/parser	Data source node (typically local playback sources)

PVMF video decoder node key strings are listed below. The key strings allow settings associated with M4v and H.263 video decoding to be queried and modified when PVMF Video Decoder node is used to decode video bitstreams to YUV.

Key Strings With Value Type	Description
x-pvmf/video/decoder/postproc_enable;valtype=bool	Flag to enable/disable postprocessing in video decoder
x-pvmf/video/decoder/postproc_type;valtype=bitarray32	If postprocessing is enabled, the postprocessing types enabled.
x-pvmf/video/decoder/h263/maxbitstreamframesize;valtype=uint32	The maximum frame size of the H.263 bitstream in bytes
x-pvmf/video/decoder/m4v/maxbitstreamframesize;valtype=uint32	The maximum frame size of the M4v bitstream in bytes.
x-pvmf/video/decoder/h263/maxdimension;valtype=range_uint32	The maximum supported dimension for H.263 bitstream. Min is width, max is height; both in pixels.
x-pvmf/video/decoder/m4v/maxdimension;valtype=range_uint32	The maximum supported dimension for M4v bitstream. Min is width, max is height; both in pixels

PVMF streaming and download source node key strings are listed below.

Key Strings With Value Type	Description
x-pvmf/net/delay;valtype=uint32	Specifies the jitter buffer duration in milliseconds (typically used in streaming sessions)
x-pvmf/net/jitterBufferNumResize;valtype=uint32	Specifies the number of times a growth in jitter buffer ought to be allowed.
x-pvmf/net/ jitterBufferResizeSize;valtype=uint32	Specifies the amount by which each jitter buffer growth happens.
x-pvmf/net/user-agent;valtype=wchar*	Specifies the user agent string in unicode
x-pvmf/net/keep-alive-interval;valtype=uint32	Specifies the keep-alive-interval in milliseconds (this is the frequency at which the player would send keep-alive notifications to the server)
x-pvmf/net/keep-alive-during-play;valtype=bool	Specifies whether keep-alive notifications need be sent during playback (typically keep-alive notifications are sent in a paused state)
x-pvmf/net/speed;valtype=uint32	Specifies the speed at which a streaming session ought to be requested from the server . Not currently supported for all streaming protocols.
x-pvmf/net/http-version;valtype=char*	Specifies the HTTP Protocol Version to be used during download / streaming.

x-pvmf/net/num-redirect-attempts;valtype=uint32 Optional params on key: 1) The key can contain a "mode=" parameter to indicate if this redirect attempts applies to streaming or download session or DLA.	Specifies the maximum number of times the client would process and act on a redirect notification from the server.
x-pvmf/net/protocol-extension-header;valtype=char* Optional params on key: 2) The key can contain "purge-on-redirect". This means that this protocol-extension-header will not be sent to the server in case of redirect. Example: "x-pvmf/net/protocol-extension-header;valtype=char*;mode=streaming;purge-on-redirect" 3) The key can contain a "mode=" parameter to indicate if this extension header applies to streaming or download session or DRM. Format of the value string: 1) The extension header is provided a s key-value pair. 2) The value string can contain an additional "method=" argument. This is used to specify the protocol methods to which this extension header applies. For example: "key=PVPlayerCoreEngineTest;value=Test;method=GET, POST"	Specifies any extension headers that need to be sent to the server.
x-pvmf/net/http-timeout;valtype=uint32	Specifies the HTTP timeout in seconds
x-pvmf/net/download-progress-info;valtype=char*	Specifies if the download progress is to be reported in percentage of bytes downloaded as opposed to percentage of time downloaded. If progress info in percent bytes downloaded is desired then please specify the value of "byte"; Please see Download Progress Usage detail for usage clarification.
x-pvmf/net/http-header-request-disabled;valtype=bool	During progressive download player uses the HTTP HEAD request upfront to ascertain the total file size. In case it is desired that this HEAD request must not

	be sent then this key can be used to disable the same.
x-pvmf/net/max-tcp-recv-buffer-size-download;valtype=uint32	Specifies the max buffer size to be used while doing recvs on the TCP socket, during a progressive download session.
x-pvmf/net/max-tcp-recv-buffer-size-streaming;valtype=uint32	Specifies the max buffer size to be used while doing recvs on the TCP socket, during a streaming session.
x-pvmf/net/rebuffering-threshold;valtype=uint32	Specifies the re-buffering threshold in milliseconds (typically used in streaming sessions). If the jitter buffer delay drops below this threshold, then player would enter re-buffering. This value must be less than the jitter buffer duration specified via the "delay" key string listed above.
x-pvmf/net/disable-firewall-packets;valtype=bool	In case of UDP streaming sessions, a firewall between the client and the server could block all UDP traffic. PvPlayerSDK attempts to unblock traffic using a proprietary algorithm, by default. This key can be used to turn off this feature.
x-pvmf/net/jitterbuffer-inactivity-duration;valtype=uint32	Specifies the jitter buffer inactivity duration in milliseconds (typically used in streaming sessions). If there is no incoming media for this amount of time PVPlayerSDK will end the streaming session with an inactivity timeout error

9.2.1 Download Progress Usage Detail

As discussed in Section 9.2, the application can configure the type of download progress data reported by PVPlayer using the x-pvmf/net/download-progress-info capability configuration. The application would receive download progress data when PVPlayer sends the PVMFInfoBufferingStatus event. The data can be retrieved by calling GetLocalBuffer() on the PVAsyncInformationalEvent object provided to the HandleInformationalEvent() callback. The progress data can be interpreted in the following three ways:

- If no content length is received from the server, the progress data is always the total number of bytes received from the server, regardless of x-pvmf/net/download-progress-info setting.
- If content length is received from the server, the progress data is by default the percentage of time duration of the clip that has been downloaded. For example, if 6 seconds of media data has been downloaded for a 30 second clip, the progress data would be 20%.

- If the application configures x-pvmf/net/download-progress-info setting to report progress in bytes, the progress data is the number of bytes downloaded divided by the total number of bytes in the file to be downloaded. For example, if 60KB of data has been downloaded for a 300KB clip, the progress data would be 20%.

9.3 Usage examples

The sequence diagram below illustrates how the application can retrieve the capability-and-configuration interface from PVPlayer engine and perform queries and changing of playback settings at the engine level and node level. Since PVPlayer engine does not support a PvmiMIOSession for the capability-and-configuration interface, NULL is passed in for interface methods with a PvmiMIOSession parameter. Also context parameter is not supported so the PvmiCapContext parameter is ignored by interface methods.

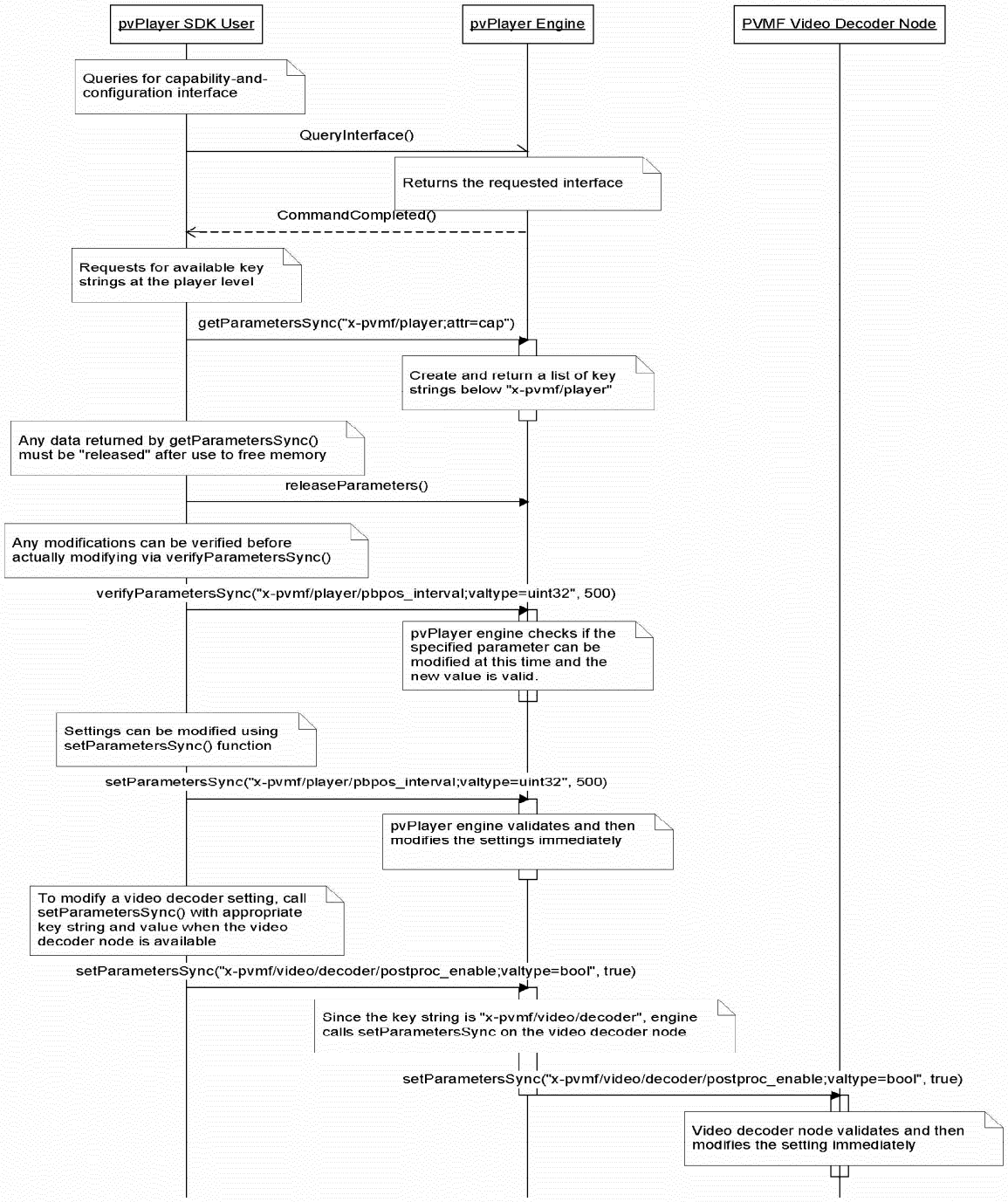

Figure 10: Capability and Configuration Interface Usage Sequence

10 Metadata Handling

Metadata is information about the multimedia data, which is not the media data itself. Across the different content types supported by PVPlayer, there are several different schemes defined for storing information. For example, the following is a short list of some of the metadata schemes that may be encountered in the supported file types: ID3v1, ID3v1.1, ID3v2, PV's metadata storage within an MPEG4 file, 3GPP Release 6 asset information within an MPEG4 file, and Apple iTunes metadata within an MPEG4 file. Typical information in the metadata includes such things as title, author, description, copyright, etc. The PVPlayer SDK supports retrieval of metadata by the relaying metadata queries to the underlying components that implement the actual parsing of the different metadata storage schemes.

10.1 Metadata retrieval APIs

Within the PVPlayerInterface, metadata is handled as key-value pairs. The APIs provide a way to obtain the list of available key strings through GetMetadataKeys() and a way to obtain the values associated with a list of keys through GetMetadataValues(). Since there are usually several metadata values, the APIs use list structures for the keys and values. Also, the lists of values can be arbitrarily long, so the APIs allow segments of any size to be retrieved with each call so that it is not required to hold the entire list at once.

10.1.1 Metadata Related Events

In certain non-local playback scenarios metadata is not readily available with the engine. Hence it could not be retrieved at the beginning of playback. In such scenarios metadata shall be fetched on the basis of informational events sent by engine.

Typical metadata related events sent by engine:

Error Code	Meaning
PVMFInfoDurationAvailable	Duration is available, and can be retrieved now. This event itself carries the duration and there's no need to issue GetMetadataValues() api to get the duration value.
PVMFInfoMetadataAvailable	Metada is ready, and application can retrieve meta data now.

10.2 Retrieving Metadata List

The PVPlayer engine relies on PVMF nodes to provide a list of metadata keys and values. Most metadata typically come from data source and parser nodes, but metadata could also come from processing nodes like decoders. The engine determines if a node supports metadata by requesting each node for the metadata retrieval extension interface. The engine retrieves the node's metadata key list by calling the GetNodeMetadataKeys() API for the node. In response to that command, the node returns information on all available metadata in the node at that moment as an array of metadata key strings. The engine concatenates the metadata key list from each node to form the overall list of keys which is provided when the GetMetadataKeys() API of the PVPlayer SDK is called. The diagram below shows how the PVPlayer engine's metadata key list is generated from each node's metadata key list including the mapping for illustrative purposes.

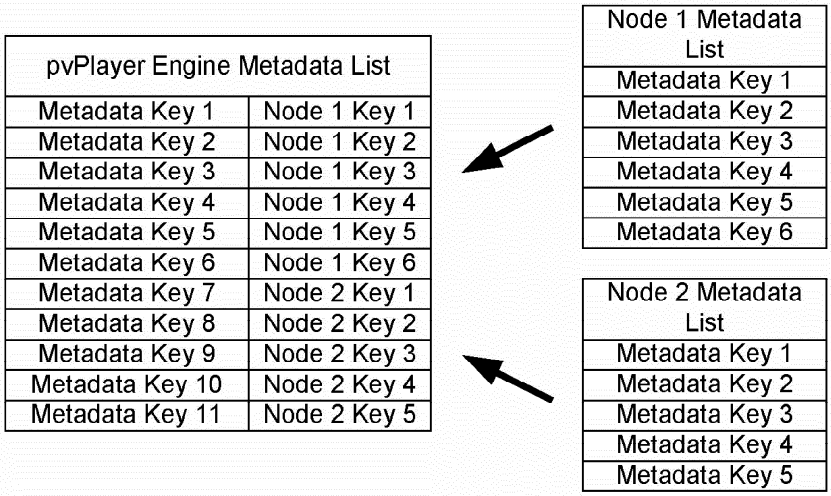

Figure 11: Mapping of Multiple Metadata Key Lists

The list of metadata keys from PVPlayer engine is dynamic and changes as nodes being used are added and removed from the data-flow graph and change states. When a node is added to the data-flow graph, new metadata keys may be added to the overall list, and when a node is removed, metadata entries may be removed. Therefore the metadata key list returned by the PVPlayer engine would only be valid for a finite amount of time.

10.3 Querying Metadata

When GetMetadataValues() is called, PVPlayer engine calls GetNodeMetadataValues() for each node that provides the metadata retrieval extension interface. As each node returns the list of requested metadata values, the metadata values are copied to the metadata value list passed in by the user of PVPlayer SDK. When all nodes return requested the metadata values, PVPlayer engine reports GetMetadataValues command as complete.

10.4 Metadata Storage

Metadata is a key-value pair where the metadata key is stored as a string while the value is often a string but may be other types such as an integer, etc. The metadata value could be one of many data types so it is stored as an union of various data types. When GetMetadataValues() is called, PVPlayer engine returns the requested values as a vector of the key-value pair structure which contains the key string along with a union containing the value. The "valtype" parameter in the returned key string specifies which of the union members to access to read the associated value. The user is responsible for parsing the key string for this "valtype" parameter to determine the data type of the value. For example a returned key string of

```
author;valtype=wchar*
```

would indicate that the returned author value is a wide-character string and is accessible in the union member corresponding to the wide character pointer.

If the metadata value is a pointer type, PVPlayer engine will allocate and deallocate the memory referenced by the pointer. But the pointer is only valid during the callback handler (i.e. CommandCompleted() function) that notifies the user GetMetadataValues command has completed. After the callback handler returns, the memory may be freed at any time and the pointer would become invalid. *Therefore, all data of interest must be copied to another storage area before the callback handler returns.*

For variable size value types (e.g. pointers such char*), the length field of the key-value pair provides the size of the valid data and the capacity field provides the total size of the buffer. Both length and capacity sizes are in the units of the type. For character string values, the length field includes the NULL terminator (e.g. if value string is "abc\0", the string is 3 characters long but length field in the key-value pair is 4). For fixed size value types, length and capacity are undefined and should not be used.

10.5 Metadata Keys

The different metadata schemes have variations on the exact set of information provided. Some consist of small fixed sets of information, while others like ID3v2 are extensible to allow new keys and new information in the future. However, there is a fair amount of similarity in the core set information provided by these different schemes. Therefore the PVPlayer SDK defines a set of simple common metadata keys that across the different content formats and metadata schemes. Internally, the appropriate metadata entry will be mapped to the appropriate common key. Other metadata may be accessible beyond the common set, but access to those values will use keys specific to the metadata scheme. The table below lists the set of simple common metadata keys. For example, the SDK user could query with the metadata key “author” to get the author information regardless of content type. It’s not guaranteed that a particular piece of content has any of this information stored in metadata. However the SDK user can be sure that if, for example, authoring information is stored in a supported metadata scheme, a query using the “author” will retrieve it.

Many of the keys are simple one-word strings, but the format of the key allows for more complicated forms, which may include optional parameter qualifiers. The syntax of the key strings follows a similar format to the PvXms extended MIME strings used of configuration and capability exchange within the framework, but there are some differences. For example, the key strings can consist of a single word (i.e., does not require at least two levels of type strings). The syntax of the metadata key is defined as follows:

```
Metadata key := root-key-string *("/" sub-key-string)
          *(";" parameter)
          ; Matching of key-string
          ; is ALWAYS case-insensitive.
       ; There MUST ALWAYS be a root-key-string

root-key-string := token

sub-key-string := token

parameter := attribute "=" value

attribute := token
            ; Matching of attributes
            ; is ALWAYS case-insensitive.
```

```
Value := token / quoted-string

token := 1*<any (US-ASCII) CHAR except SPACE, CTLs,
           or tspecials>

tspecials :=  "(" / ")" / "<" / ">" / "@" /
              "," / ";" / ":" / "\" / <">
              "/" / "[" / "]" / "?" / "="
              ; Must be in quoted-string,
              ; to use within parameter values

quoted-string = <"> *(qtext/quoted-pair) <">; Regular qtext or
                                           ;   quoted chars.

Qtext       =  <any CHAR excepting <">,     ; => may be folded
               "\" & CR, and including
               linear-white-space>

quoted-pair =  "\" CHAR                     ; may quote any char

                                              octal    decimal
CHAR        =  <any ASCII character>        ; (  0-177,  0.-127.)
```

When the values are returned from a query the key will include some additional parameters providing further information about how to interpret the value. For example, the key returned from a query for author may look like:

```
author;valtype=whcar*
```

which indicates the value of the author string can be found in the wide character array member of the key-value pair structure. The valtype parameter will be included with every returned key since it is necessary to specify the member of the key-value pair structure to use.

Key string	Description	Notes
album	Album name	Value is typically a null-terminated string (either narrow or wide character).
artist	Artist or performer	Value is typically a null-terminated string (either narrow or wide character).
author	Author or writer	Value is typically a null-terminated string (either narrow or wide character).
classification	Classification	Value is typically a null-terminated string (either narrow or wide character).
clip-type	High-level classification of clip describing whether it is local, streaming, download, etc.	Value is a null-terminated character string. Defined values include: local, streaming, download, fasttrack.
comment	Comment string	Value is typically a null-terminated string (either narrow or wide character).
compilation	Typically found in music content.	The value is a null-terminated string (either narrow or wide character). There is usually also

		a language code.
composer	Music composer	Value is typically a null-terminated string (either narrow or wide character).
copyright	Copyright holder	Value is typically a null-terminated string (either narrow or wide character).
date	Creation date	The date value will be returned as a string represented in a subset of ISO 8601 format. For example, “”yyyy”, “yyyy-mm”, etc where yyyy represents the year and mm the month. See the ID3 specification or reference [2] for other possible examples and more details.
description	Brief description of the content	Value is typically a null-terminated string (either narrow or wide character).
duration	Duration / length of the clip	Could be returned as an integer representing the duration along with a timescale or the string “unknown” if the duration is not known. See description below for more details.
duration-from-metadata	Duration of the clip provided in it’s metadata	Value is an unsigned 32-bit integer representing the duration along with a timescale.
genre	Genre	The value will typically be an integer code or string. See description below for details.
graphic	Location of an associated graphic or the actual graphic.	Value is typically a null-terminated string (either narrow or wide character) or an attached picture using a format like the ID3 attached picture format.
id	Product ID / SKU / unique ID	No specific standard defined for this field, so it would typically be returned as a string.
keyword	Content specific keyword(s)	The value is a null-terminated string (either narrow or wide character). There is usually also a language code.
lyricist	Lyricist. Typically found in music content.	The value is a null-terminated string (either narrow or wide character). There is usually also a language code.
lyrics	A simple string containing the words spoken or sung within the song.	The value is a null-terminated string (either narrow or wide character). There is usually also a language code.
music-selling-agency	Typically found in music content.	The value is a null-terminated string (either narrow or wide character). There is usually also a language code.
music-label	Typically found in music content.	The value is a null-terminated string (either narrow or wide character). There is usually also a language code.
music-rights-holder	Typically found in music content. This could be different from copyright.	The value is a null-terminated string (either narrow or wide character). There is usually also a language code.
music-rights-information	Typically found in music content.	The value is a null-terminated string (either narrow or wide character). There is usually also a language code.
music-url	Typically found in music content.	The value is a null-terminated string (either narrow or wide character).