performer	Performer. Typically found in music content.	The value is a null-terminated string (either narrow or wide character). There is usually also a language code.
playlist	Playlist name	Value is typically a null-terminated string (either narrow or wide character).
podcast-url	Podcast URL	The value is a null-terminated string (either narrow or wide character).
purchase-date	Content purchase 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.
rating	Rating information	There may be several different
src	Clip source / filename	Value is typically a null-terminated string (either narrow or wide character).
title	Title or name of the clip	Value is typically a null-terminated string (either narrow or wide character).
num-tracks	The number of tracks in the clip	Value is typically a 32-bit unsigned integer
version	Software version of the authoring software	Value is typically a null-terminated string (either narrow or wide character).
year	Year of recording/performance. Typically applies to music files.	Value is typically a 32-bit unsigned integer

10.5.1 Limiting the Metadata Value Size

In certain cases it may be desirable to specify the maximum size of the metadata value that is being requested. This way the application can have some control over the amount of memory that will be used to return the metadata value. Since the metadata may include items like graphics (e.g., album art, etc), which can be fairly large, it is important for applications to have enough control to avoid out of memory conditions in memory-constrained situations.

The **maxsize** parameter in the request key string is used to specify the maximum size, in bytes, that should be returned for the requested value. It is an optional parameter that may be applied to any *variable-length* metadata value (i.e., strings, arrays, etc). The maxsize parameter does not apply to values that are returned as fixed-sized elements of the PvmiKvp union (e.g., int32, uint8, etc). The reason that it only applies to the variable-length values is that the PvmiKvp structure needs to be provided in every case to return the key value and report the required maximum size. Also the intention of the maxsize parameter is mainly to provide a way to deal with large metadata values. In case of metadata values that are strings, maxsize parameter will be interpreted to mean maximum size, in number of characters not including the NULL terminator, and not maximum size, in bytes.

With a maxsize parameter defined, there is the question of the behavior in the case that the maxsize is exceeded. Either the value could be returned truncated to the specified size or the information about the required size could be returned without the actual value (i.e., no space would be allocated to hold the value and no portion of the value be returned). Truncation is reasonable where an incomplete part of the

value is potentially meaningful and useful (e.g., a string value). For cases where the value is really only useful when it can be returned in its entirety (e.g., a graphic image), then it does not make sense to truncate the value. Instead the request should be answered by indicating the required amount of memory to retrieve the complete value. Since the majority of the common metadata values are strings, the default behavior in the case where the value size exceeds the specified maxsize parameter will be to return the truncated value. However, an optional boolean parameter called ***truncate*** can be specified to indicate the desired truncation behavior. For example,

```
title;maxsize=100;truncate=false
```

indicates that the returned title value should have a max size of 100 bytes, and if the actual size exceeds that length, only the information about the required size should be returned (i.e., no truncation should happen). By contrast, the request

```
title;maxsize=100;truncate=true
```

differs by the fact that the title value should be truncated to at most 100 bytes. Note that the request with

```
title;maxsize=100
```

is equivalent to the one with the *truncate=true* parameter since truncation is the default behavior. ***Note that strings consisting of multibyte characters (e.g., UCS-2, UTF-8, etc) will be truncated to a whole number of characters that is less than or equal to the specified number of bytes.***

If the metadata value is larger than the specified maxsize, the required number of bytes is returned in the ***reqsize*** parameter. The required size is returned regardless of whether the value is truncated and returned or not. For example, if the request for the title is truncated at 100 bytes but the actual size of the string is 137, then the returned key string would look like:

```
title;valtype=wchar*;reqsize=137
```

The reqsize parameter is only included in cases where the entire value is not returned (i.e., either when it was truncated or no part of value is returned).

10.5.2 Duration

The duration value is typically returned as an integer value and may include an optional parameter that specifies a timescale. *If no timescale is specified, then the default is milliseconds*. Some examples include

`duration;valtype=uint32` (the duration is an integer representing milliseconds)
`duration;valtype=uint32;timescale=8000` (the duration is an integer representing the duration in a timescale of 8 kHz)
`duration;valtype=char*` (the duration is returned as the string “unknown”).

The duration is not stored explicitly in all the supported file formats. File formats like mpeg4 store the duration value explicitly so it is a simple and quick matter to extract the value. Other simpler file formats like mp3, aac, and amr consist of a concatenation of encoded frames without the duration explicitly stored anywhere. Therefore, the duration must be determined by parsing the entire file, which can be computationally expensive and time-consuming in these cases. By default, the duration will be returned as unknown in these cases initially and an event will be sent once the duration has been determined as a

part of normal playback. However, it is possible to request that the duration be computed by including an additional option in request key. The form of the request key would look like:

```
duration;compute=true
```

to request that the duration is computed if necessary and possible.

The *duration-from-metadata* key is for situations where the duration is provided as an optional part of metadata and is not guaranteed or even necessarily reliable for the content type. For content that duration derived from metadata in a consistent and reliable way, the *duration* metadata key will be used. This can be queried via

```
duration-from-metadata;valtype=uint32
```

10.5.3 Genre

The genre value is often stored as an integer that corresponds to one of the values defined by ID3v1. In these situations, the returned value would be an integer and the returned key would include a qualifying parameter to indicate that it should be interpreted as an ID3v1 genre code as follows:

```
genre;valtype=uint32;format=id3v1
```

In the case of ID3v2, the classification may consist of a mixture of the ID3v1 code and an arbitrary string. In this case, the genre would be returned as a string with id3v2 indicated in the format as follows:

```
genre;valtype=wchar*;format=id3v2.
```

In some cases the genre may simply be a free-form string. In those cases the format parameter would not be provided because there is no special way of interpreting the value other than as a string. For example, it the key string would like the following:

```
genre;valtype=wchar*.
```

10.5.4 Graphic

The graphic value may be either a reference to an external image (i.e., stored separately from the media source) through a URL string or the image itself. In the case of the external reference, the information would be returned as a URL string. For example,

```
graphic;valtype=char*
```

is an example of a key string for a external reference graphic where the character string is a URL. If the graphic is returned as a character string with no other format specification, then it should be interpreted as a URL.

A popular format for directly holding the image within the media file is the ID3v2 attached picture format (i.e., the APIC frame). This same format is also used within ASF files for the WM/Picture metadata value. The format parameter for the key string indicates that the value is in the APIC format as follows:

```
graphic;format=APIC;valtype=ksv
```

The format includes the following information:

- A MIME type string describing the format of the image data
- A picture type code that classifies the content of the image
- A text description
- The binary picture data.

The picture type is one byte with values defined in the ID3v2 specification [3]. For convenience, the current table at the time this document was produced is included below, but the reference, [3], should serve as the official source of the defined values.

Code (in hex)	Description	Code (in hex)	Description
$00	Other	$0B	Composer
$01	32x32 pixels 'file icon' (PNG only)	$0C	Lyricist/text writer
$02	Other file icon	$0D	Recording Location
$03	Cover (front)	$0E	During recording
$04	Cover (back)	$0F	During performance
$05	Leaflet page	$10	Movie/video screen capture
$06	Media (e.g., label side of CD)	$11	A bright coloured fish
$07	Lead artist/lead performer/soloist	$12	Illustration
$08	Artist/performer	$13	Band/artist logotype
$09	Conductor	$14	Publisher/Studio logotype
$0A	Band/Orchestra		

In general, there can be an arbitrary number of APIC frames associated with a file, but there may only be one instance of type $01 and one instance of type $02 according to the specification. The key string syntax for these

It may be desirable to request information on the number of APIC frames in the file before actually requesting them. This information can be requested by using the key string

```
graphic/num-frames;format=APIC
```

Since there can be multiple frames in the file it may be desirable to obtain the descriptions and select the frame(s) of interest before actually requesting image data. The following key string can be used to request the multiple entries.

```
graphic/description;format=APIC;index=X…Y
```

where the values X and Y refer to the start and end index values of the requested frames (in the range 0 to num-frames-1). The structure returned is the same as the one used for the full APIC information except the binary image data buffer is empty (i.e., the description string, text encoding, mime type, and picture type, and the size of the binary image data are returned). To request the full value including the binary image data, the request key string would simply use 'graphic' string as in this example:

```
graphic;format=APIC;index=X.
```

It is also possible to restrict the query to a specific picture type by adding the "pict-type" parameter to the key string. In that case, the returned values are narrowed to the set of the frames that have the matching picture type. For example, the key string

```
graphic/num-frames;format=APIC;pict-type=0F
```

specifies that the value returned should correspond to the number of frames with picture type equal to 0F. If the pict-type parameter is applied to a request with an index parameter, then the range of valid index values is restricted to lie between 0 and num-frames-1, where num-frames is the number of frames with that picture type. For example, to get the second graphic value with picture type 0F, the request might look like

```
graphic;format=APIC;pict-type=0F;index=1.
```

The maxsize parameter is another common key string parameter that can be applied to the graphic value request as described in Section 10.5.1. Refer to the API documentation for details of the structure that is returned for the APIC format.

All images by default are considered storable/savable. However there could be cases wherein the content provided might mark some of the images as not storable. In those cases a "not-storable" string would be added to the returned key string. For example key string for a non-storable image would look like

```
graphic;format=API;index=1;not-storable
```

10.6 Track-level Information

Certain file formats, such as mpeg4, as well as streamed presentations can contain multiple media streams or tracks. The track-level information provides details at the individual media level on such things as format, sample rate, bitrate, etc. The mechanism for accessing track information will apply for all clips regardless of how many tracks are included. For simple file formats, there will only be one track while others may include an arbitrary number. The metadata key, "num-tracks," will return the number of tracks within the clip. Information for individual tracks is accessed or qualified in the returned value with the "index" parameter, which has a range from 0 to (num-tracks – 1). For example:

```
track-info/type;index=0;valtype=char*
```

would be one possible key string for the track-level type information of the first track (i.e., index 0).

10.6.1 Compact Representation of Ranges

When querying for a list of available keys from a file source with multiple tracks, it will have a set of keys for track-level information where the only difference is the index parameter. One possible way of returning the set of keys is to simply include them all individually in the list (e.g., `track-info/type;index=0, track-info/type;index=1, track-info/type;index=2, track-info/type;index=n-1`). However, a more compact representation is allowed for the index using a format for expressing a range. For example, the string

```
track-info/type;index=0…8
```

represents a set of 9 keys, 0 through 8, in a single string using the range representation for the index.

10.6.2 General Information

The table below lists general track-level information that would be available for any track.

Key string	Description
track-info/type	The type information for the media stream typically expressed as a MIME type.
track-info/track-id	The track-id is the identifier specified within the file if any. It may be different than the "index" parameter, which is simply used to iterate through the track-info metadata.
track-info/sample-rate	The sample-rate of the media in samples per second. Applicable to audio tracks. Provides the sampling rate of audio.
track-info/bit-rate	The bit-rate in bits-per-second.
track-info/duration	The track-level duration. The format is the same as the clip-level durations.
track-info/num-samples	The number of samples in the track.
track-info/selected	Boolean value that signals whether the track specified by the index is selected for playback or not.
track-info/frame-rate	Applicable only to video tracks. Provides an **approximate** estimate of the video frame rate.
track-info/codec-name	Value is typically a null-terminated string (either narrow or wide character).
track-info/codec-description	Value is typically a null-terminated string (either narrow or wide character).
track-info/codec-specific-info	The uint8 pointer provides the codec specific information.
track-info/track-number	The track-number is the identifier specified within the file if any. Typically found in music files and can be different from both "index" and "track-id" metadata fields.
track-info/max-bitrate	Maximum bit-rate in bits-per-second.

10.6.3 Format Specific Information

Some track-level information is specific to the type of media. Below are the defined video and audio track-level information.

Video-specific track-level information

track-info/video/format	Detailed video format information (e.g., profile and level information for mpeg4)
track-info/video/height	Height of the video frame.
track-info/video/width	Width of the video frame.
track-info/video/display-height	Display height of the video frame. This need not be same as the decode height.
track-info/video/display-width	Display width of the video frame. This need not be same as the decode width.

Audio-specific track-level information

track-info/audio/format	Detailed audio format information
track-info/audio/channels	Number of audio channels (e.g., 1 = mono, 2 = stereo, etc).
track-info/audio/bits-per-sample	Mainly relevant for PCM audio files.

10.7 Codec Level Format Specific Information

The codecs may also expose similar types of information, which are actually extracted from the bitstream. The codec-level information can be more reliable than the track-level information at times (e.g., in some files the height and width information has been found to be incorrect). The format-specific codec-level information is shown below

Video-specific codec-level information

codec-info/video/format	Detailed video format information (e.g., profile and level information for mpeg4)
codec-info/video/height	Height of the video frame.
codec-info/video/width	Width of the video frame.

Audio-specific codec-level information

codec-info/audio/format	Detailed audio format information
codec-info/audio/channels	Number of audio channels (e.g., 1 = mono, 2 = stereo, etc).
codec-info/audio/sample-rate	The sample-rate of the audio data in samples per second. For PCM audio, it represents the frequency in hertz.
codec-info/audio/bits-per-sample	Bits-per-sample of the output PCM

10.8 Language Codes

3GPP Release 6 defines a number of metadata elements as part of the asset information specified in the document TS 26.244v6.2.0. These metadata strings can be represented in different languages, so there is a language code associated with each entry to encode the language of the string. The language codes are stored as packed ISO-639-2/T language codes, which are basically 3 character codes assigned to each language. The table below lists a just a few examples of the languages and the associated language code, please refer to a reference on ISO-639-2/T for a complete list such [1]:

3-Letter Language Code	Description
eng	English
fre/fra	French
ger/deu	German

If the language code exists it will be returned in the iso-639-2-lang parameter. Otherwise English should be assumed. It is expected that content may contain the same metadata in multiple languages, so the language parameter in the returned key string can be used to select the value in the appropriate language based on the user preferences. An example of a key string with a language code is

```
track-info/type;index=0;valtype=wchar*;iso-639-2-lang=ger
```

10.9 DRM Related Metadata

There are a number of metadata values related to license information for content protected with some form of digital rights management (DRM). For a particular piece of content, not all the values in the table will be available. This set of metadata provides information that describes the issuer of the license, which operations are allowed, when it expires, etc. Note that certain time-based licenses may have only start times, only expiration times, or both start and expiration times. Values will only be returned if the license has a corresponding value for that key string, so for example, if the license only has a start time then queries for the license-expiry would not return a value.

Key string	Description	Notes
drm/is-protected	Absence of value indicates that the content is unprotected; however if value returned; indicates whether the content is DRM- protected (true) or DRM-unprotected (false).	Value is bool type when provided.
drm/license-issuer	This is the URI of the license issuer.	Value returned in the char* type.
drm/allowed-usage	Provides information on the approved usage of the content. The returned value is a packed bit array with the possible permission values.	Packed bit array. Possible values include: Play, Pause, Resume, Seek forward, Seek backwards, stop, print, download, save, execute, preview.
drm/is-license-available	True/false value indicating whether the license is available.	Value returned in the bool type.
drm/auto-acquire	True/false value indicating whether there will be an attempt to automatically acquire a new license when necessary.	Value returned in the bool type.
drm/license-type	License types fall into following categories: ● time based (has an start and end time) ● duration based (a certain amount of time since first use) ● count based ● or a combination of count and one of time-related types ● unlimited (no limit on the counted license)	Value returned would be a string that will take any of ● the following forms: ● "time", "duration", "count", "time-count", "duration-count", "unlimited"

drm/num-counts	Counts remaining	Value is returned as a uint32
drm/license-start	The starting time for the licensed interval	All start and end times are in ISO 8601 Timeformat ● The format is as follows. Exactly the components shown here must be present, ● with exactly this punctuation. Note that the "T" appears literally in the string to indicate the beginning of the time element, as specified in ISO 8601. Year: YYYY (e.g., 1997) Year and month: YYYY-MM (e.g., 1997-07) Complete date: YYYY-MM-DD (e.g., 1997-07-16) Complete date plus hours and minutes: YYYY-MM-DDThh:mmTZD (e.g., 1997-07-16T19:20+01:00) Complete date plus hours, minutes and seconds: YYYY-MM-DDThh:mm:ssTZD (e.g., 1997-07-16T19:20:30+01:00) Complete date plus hours, minutes, seconds and a decimal fraction of a second: YYYY-MM-DDThh:mm:ss.sTZD (e.g., 1997-07-16T19:20:30.45+01:00) where: YYYY = four-digit year MM = two-digit month (01=January, etc.) DD = two-digit day of month (01 through 31) hh = two digits of hour (00 through 23) (am/pm NOT allowed) mm = two digits of minute (00 through 59) ss = two digits of second (00 through 59) s = one or more digits representing a decimal fraction of a second TZD = time zone designator (Z or +hh:mm or –hh:mm) This profile defines two ways of handling time zone offsets: ● Times are expressed in UTC (Coordinated Universal Time), with a special UTC designator ("Z").

		• Times are expressed in local time, together with a time zone offset in hours and minutes. A time zone offset of "+hh:mm" indicates that the date/time uses a local time zone which is "hh" hours and "mm" minutes ahead of UTC. A time zone offset of "-hh:mm" indicates that the date/time uses a local time zone which is "hh" hours and "mm" minutes behind UTC. For example: 1994-11-05T08:15:30-05:00 corresponds to November 5, 1994, 8:15:30 am, US Eastern Standard Time. 1994-11-05T13:15:30Z corresponds to the same instant. If the start time is not set then the value should be interpreted as "now".
drm/license-expiry	End time of licensed interval.	See previous description of the time format. A query for this key will not be answered if there is no specified end time.
drm/duration	Duration of the license specified in number of seconds.	The value will be returned as a uint32.
drm/license-store-time	The time when the license was added to the license store.	See previous description of the time format. A query for this key will not be answered if there is no support for looking up the time that the storage for a particular license store.
drm/dla-data	In cases where direct (over-the-air) license acquisition is expected, this field will contain opaque data required by the AcquireLicense command.	The value will be returned as a uint8*. The number of bytes will be in the Kvp length field.
drm/is-forward-locked	True/false value indicating whether the content is forward locked.	Value returned in the bool type.

10.10 Access to Other Metadata

Depending on the content format being accessed and the metadata storage scheme, there may be additional metadata entries that do not fall within the list of values described above. This situation is especially true for extensible metadata schemes like ID3v2. The parser used by the engine may not necessarily understand how to interpret the data in the metadata frame, but it can provide the raw data back to application for it to interpret. The form of the keystrings for requesting ID3v2 frames is:

```
id3v2/<four-character frame ID>
```

where <four-character frame ID> is the four-character code defined by the ID3 specification. If the key string is present, the returned value will include the ID3 version, the frame ID, frame size, frame flags, and the raw data contained in the frame. See the API documents for the exact definition of the id3v2 frame structure. This id3v2 frame structure will be returned in the key-specific value field of the returned key-value pair structure.

10.11 Receiving Metadata from Informational Event Callback

For server side playlist streaming sessions, PVPlayer engine also sends an unsolicited information event — PVMFInfoPlayListClipTransition. A playlist contains several playlist elements. When the client gets notified about the transition to a new playlist element, the player engine sends this event. It should NOT be used as an accurate indication of the transition point on UI because of the delay like jitter etc. This event also carries the extra meta data about the next playlist element. The event data is a PVMFRTSPClientEngineNodePlaylistInfoType struct:

```
typedef struct
{
 uint32 iPlaylistUrlLen;
 char *iPlaylistUrlPtr;
 uint32 iPlaylistIndex;
 uint32 iPlaylistOffsetSec;
 uint32 iPlaylistOffsetMillsec;

 uint32 iPlaylistNPTSec;
 uint32 iPlaylistNPTMillsec;

 //max 256
 uint32      iPlaylistMediaNameLen;
 char iPlaylistMediaNamePtr[256];

 //max 512
 uint32      iPlaylistUserDataLen;
 char iPlaylistUserDataPtr[512];
}PVMFRTSPClientEngineNodePlaylistInfoType;
```

10.12 Metadata Retrieval Usage Example

To illustrate how metadata list is generated and returned to the user of PVPlayer SDK and how metadata value is returned, a sequence diagram between the user of PVPlayer SDK, PVPlayer engine, and two PVMF nodes that support metadata retrieval is shown below.

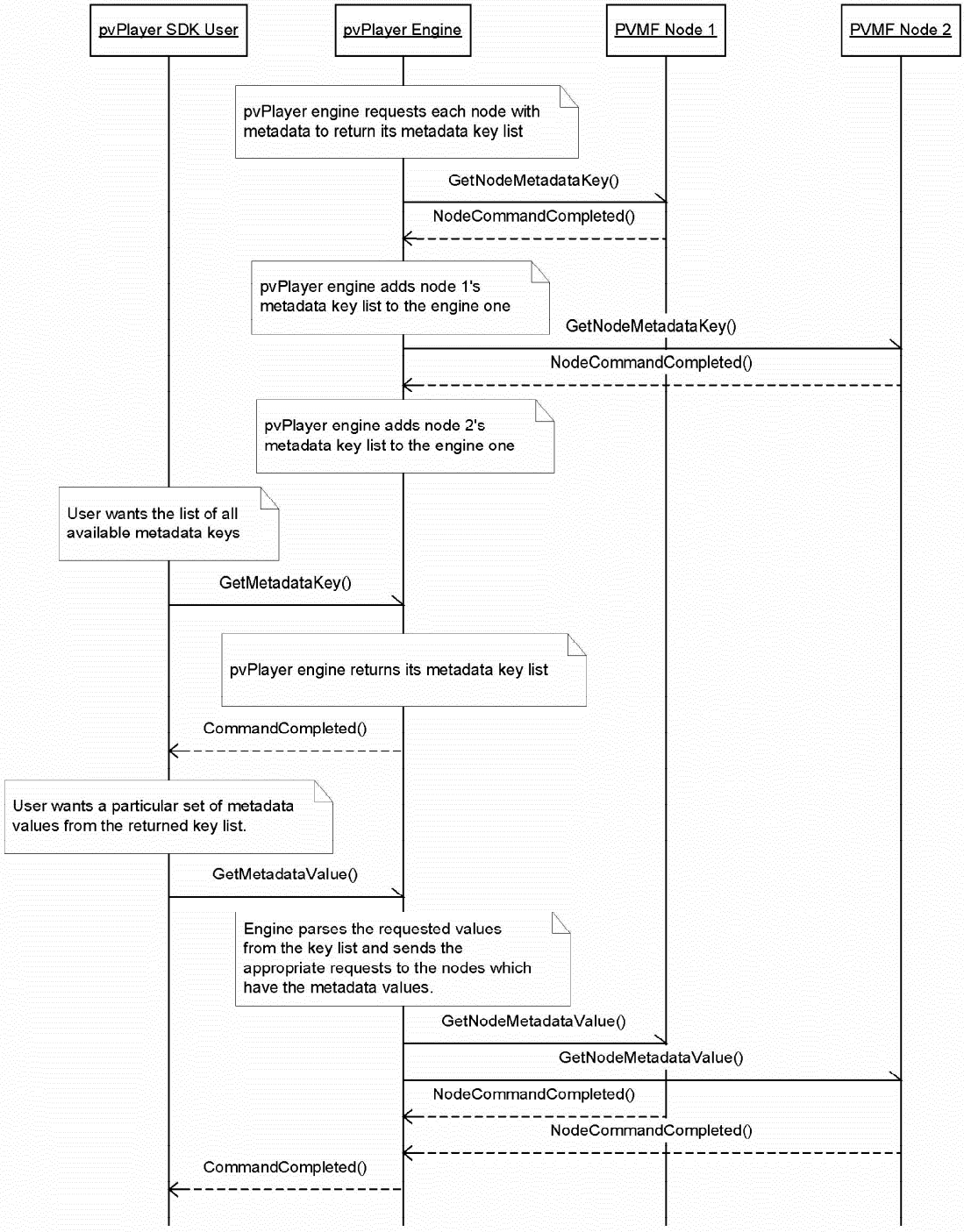

Figure 12: Metadata Retrieval Usage Sequence

10.13 Supported Key Strings in Select PVMF Nodes

The table below lists the supported metadata key strings in several PVMF nodes. The key string list is the comprehensive list, but actual key list could be a subset depending on the information available in the data source.

PVMF Node	Supported Key Strings
PVMFMP4FFParserNode	author title description rating copyright version date duration num-tracks track-info/type track-info/track-id track-info/duration track-info/bit-rate track-info/audio/format track-info/video/format track-info/video/width track-info/video/height track-info/sample-rate
PVMFMP3FFParserNode	title artist album year comment copyright genre tracknumber num-tracks duration track-info/bit-rate track-info/sample-rate track-info/audio/format track-info/audio/channels
PVMFAACFFParserNode	title artist album year comment copyright genre tracknumber num-tracks duration track-info/bit-rate

	track-info/sample-rate track-info/audio/format
PVMFAMRFFParserNode	duration num-tracks track-info/bit-rate track-info/audio/format
PVMFWAVFFParserNode	duration num-tracks track-info/bit-rate track-info/sample-rate track-info/audio/format track-info/audio/channels track-info/audio/bits-per-sample
PVMFVideoDecNode	codec-info/video/format codec-info/video/width codec-info/video/height
PVMFAVCDecNode	codec-info/video/format codec-info/video/width codec-info/video/height
PVMFAACDecNode	codec-info/audio/format codec-info/audio/channels codec-info/audio/sample-rate
PVMFWMADecNode	codec-info/audio/format codec-info/audio/channels codec-info/audio/sample-rate
PVMFWMVDecNode	codec-info/video/format codec-info/video/width codec-info/video/height

11 Playback Position

PVPlayer engine provides the application with methods to obtain the current playback position of the media being played. The application can use the playback position data as strictly informational data to display to the user or to make decisions during media playback (e.g. pause playback 5 seconds into playback).

The position can be retrieved by having the application make API calls or PVPlayer engine can send the playback position periodically via the unsolicited informational event callback. For both methods, the application can change the playback position units from the default of millisecond time unit.

11.1 Retrieve Playback Position Using API Call

PVPlayer SDK provides two API to retrieve the current playback position from PVPlayer engine: GetCurrentPosition() and GetCurrentPositionSync(). Both APIs perform the same function but the latter completes the request synchronously instead of asynchronously. In both APIs, the user must provide a reference to a PVPPlaybackPosition object which is an input/output parameter. The input parameter portion is the iPosUnit field which allows the user to request the units to use for the playback position. The default units is in milliseconds but the user can request the position in other time units such as seconds, hours, and SMPTE time code, or non-time units such as percentage of whole clip, sample number, and offset from beginning of the file in bytes. Availability of playback position in non-time units would depend on the support from the underlying nodes and source media being used. If non-time units is not supported, these APIs will return with PVMFErrNotSupported error code.

11.2 Receive Playback Position from Informational Event

PVPlayer engine also sends the current playback position periodically as an unsolicited informational event with PVMFInfoPositionStatus event code and player specific event code of PVPlayerInfoPlaybackPositionStatus (=8193) in PVPlayerErrorInfoEventTypesUUID event code space (=0x46fca5ac, 0x5b57, 0x4cc2, 0x82, 0xc3, 0x03, 0x10, 0x60, 0xb7, 0xb5, 0x98). The position value is stored in the local data buffer of the informational event. The application is responsible for "listening" for this event in the informational event callback handler if it wants to obtain the current playback position by this method.

The position units and the time length of the reporting period can be queried and modified via the capability-and-configuration extension interface of PVPlayer engine. The default settings are milliseconds for playback position units and 1000 milliseconds for the reporting period. For more information on how to query and modify these settings via the capability-and-configuration interface, refer to the Capability Query and Configuring Settings section. Support for non-time position units would change based on the underlying nodes and source media being used. Therefore, if support for non-time position units becomes unavailable, PVPlayer engine will automatically change to the default of milliseconds.

12 Frame and Metadata Utility

A common use-case for player functionality involves retrieving the metadata information along with a frame from the video stream to be used as a thumbnail or other still image representation of the clip. For example there may be gallery view of the available content stored on the filesystem, which is presented to the user as a still image frame from each clip along with some metadata information such as title, author, etc. The player engine APIs can certainly be used directly to obtain the necessary information. However the PVFrameAndMetadataUtility simplifies the task for the application by hiding some of the interaction with the player engine for this use-case.

12.1 Creating and Deleting the Utility

Instances of the PVFrameAndMetadataUtility are created and deleted using static member functions of the factory class. The factory function used to produce a new instance of the utility class takes a MIME string argument, which specifies the desired output format for the video frame, as well as references to observer classes for receiving callbacks from the utility. Internally, the utility creates an instance of the player engine. The diagrams below show sequences for creating and deleting the utility instance.

The format of the video frame that will be returned in the GetFrame calls is specified as an argument to the factory function when creating an instance of the utility class. A MIME string is used to specify whether the frame should be YUV420, RGB16, etc. The header file pvmf_format_types.h contains a listing of many of the common MIME strings for the different video frame formats. If the output format cannot be supported for a given input source that is specified later, then an error will be returned from the GetFrame call.

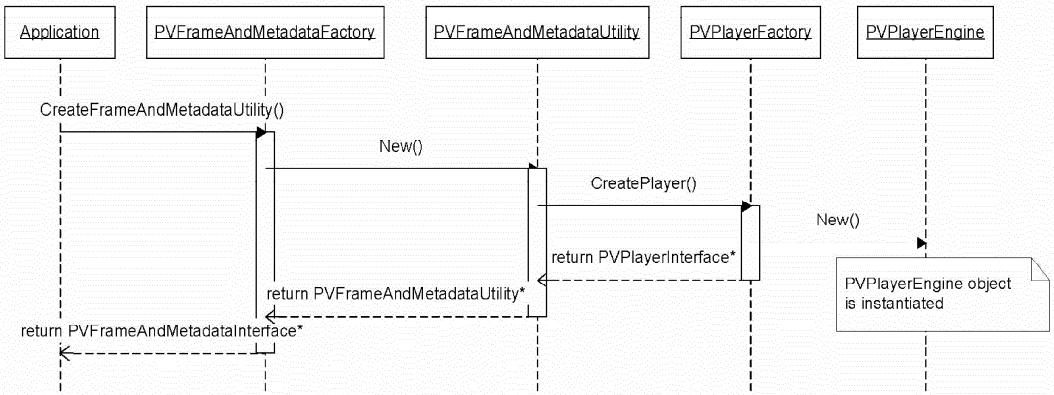

Figure 13: Create the Utility

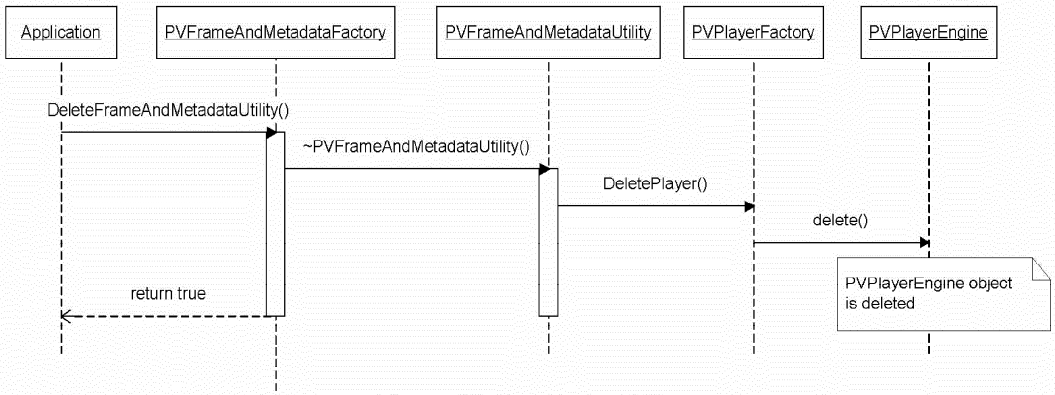

Figure 14: Delete the Utility

12.2 Options for Specifying the Desired Frame

The GetFrame() API is used to retrieve a frame specified in the frame selector argument. There are a few options for specifying the desired frame:

- the exact frame index with 0 corresponding to the first frame,
- the time offset of the frame,

These two options are used to select a specific frame based on either the frame index or the time offset of the frame. An example where this type of specification might be used is for creation of a thumbnail image from the first frame. The PVFrameSelector data type is used to hold the information on the desired frame.

In many cases, the first frame of the clip may not contain a meaningful image (e.g., the first frame may be a black frame). Therefore, another alternative is to let the Utility use an internal algorithm to autodetect a frame of interest. To achieve this, the user of the utility has to set the source context data with the BITMASK_PVMF_SOURCE_INTENT_THUMBNAILS intent. The following is an example of how it should be used.

```
// create the source context data for autodetection of thumbnails
iSourceContextData = new PVMFSourceContextData();
iSourceContextData->EnableCommonSourceContext();
//set the intent to thumbnails
 iSourceContextData->CommonData()->iIntent =
                        BITMASK_PVMF_SOURCE_INTENT_THUMBNAILS;

iDataSource->SetDataSourceContextData((OsclAny*)iSourceContextData);

iDataSource->SetDataSourceURL(wFileName);
iDataSource->SetDataSourceFormatType(iFileType);
OSCL_TRY(error, iCurrentCmdId=iFrameMetadataUtil->AddDataSource(*iDataSource,
                                                    (OsclAny*)&iContextObject));
```

12.3 Set Timeout for Frame Retrieval

The default timeout set for the frame retrieval is 30 seconds. The user of the utility has the option to alter the value of this timeout. This can be achieved by querying for the extension interface PvmiCapabilityAndConfig via the API QueryInterface(). The pointer to the interface obtained provides the flexibility to the user to set the timeout using the following KVP:

```
x-pvmf/fmu/timeout-frameretrieval-in-seconds;valtype=uint32
```

12.4 Usage Sequence

The main sequence for interfacing with the PVFrameAndMetadataUtility is shown in the figure below. As the diagram shows, the utility takes care of some of the steps of interaction with the player engine in order to get a specific frame or retrieve the metadata. The metadata is available to the application after the completion of the AddDataSource call to the utility. The AddDataSource, Init, AddDataSink, Prepare, Start, and Pause calls to the player engine are all hidden inside the processing of this request. The player engine is taken to a paused playback state to allow the datapath to be created and to allow the user to retrieve metadata from nodes within the datapath (e.g. codec information from decoder nodes).
There are two variants of the GetFrame call, which allow the frame buffer to either be provided by the application or the utility. The diagram below shows the case where the buffer is provided by the utility, in which case it must be returned once it is no longer needed using the ReturnBuffer call.

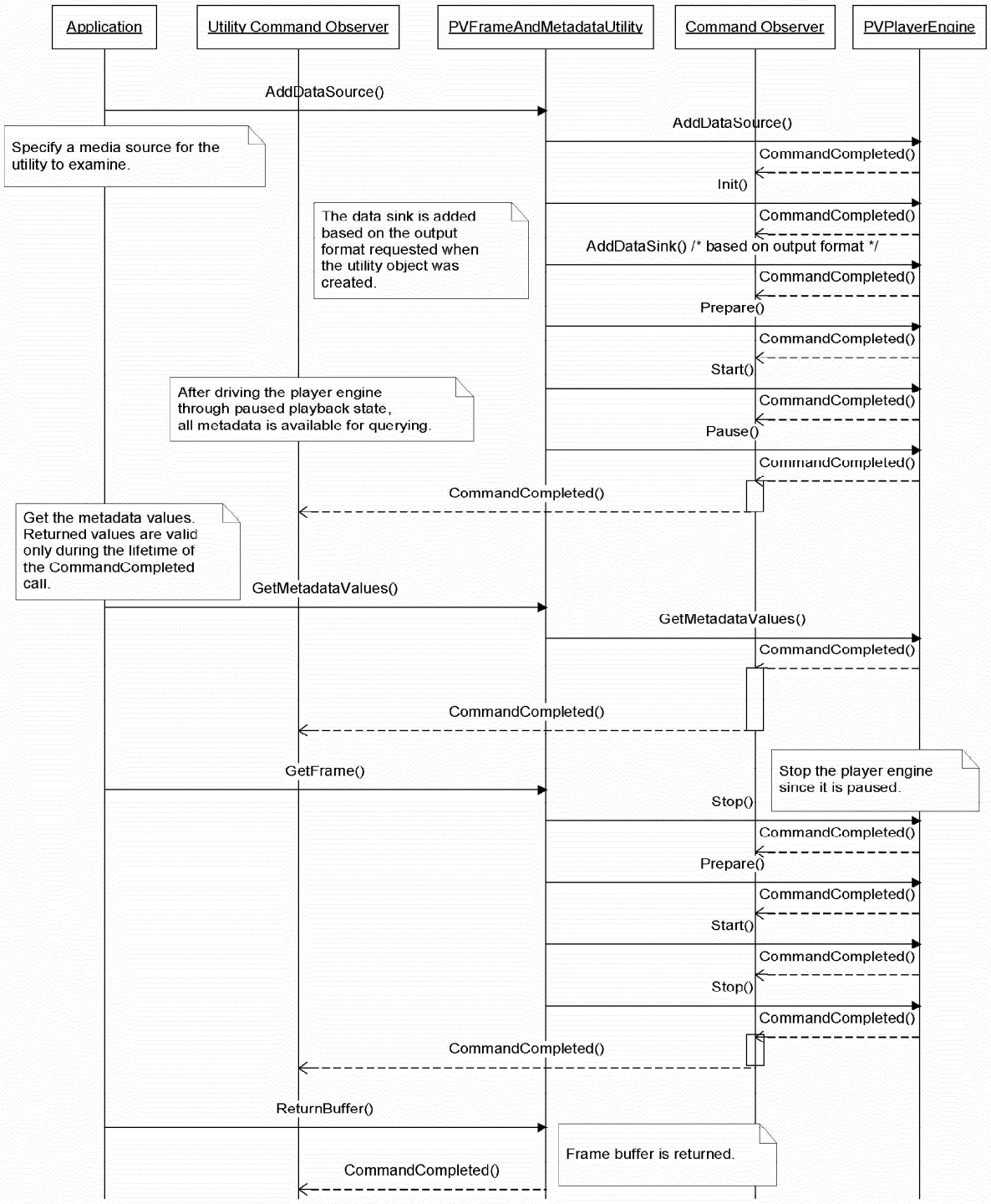

Figure 15: Frame and Metadata Utility Usage Sequence

13 Error and Fault Handling

13.1 Error Handling

Error is an erroneous system behavior that deviates from the design specifications. PVPlayer SDK will detect and handle any errors reported within its components or outside components (e.g. platform services, platform specific decoders). Based on the type of error, PVPlayer SDK will decide whether to report the error to the user of the SDK or not and whether to handle the error before continuing on. The reporting mechanism would depend on the interface between PVPlayer SDK and its user. With the OSCL-based interface, PVPlayer SDK reports errors via the command completion callback if the error occurs during an PVPlayer SDK API command processing or via the observer callback, PVErrorEventObserver, if the error is an unsolicited event.

The following section provides an overview of error types detected in PVPlayer SDK and error message reported by PVPlayer engine. Depending on the platform and PVPlayer SDK configuration, the list of error messages could be larger or smaller. For information on error events on a particular platform, refer to the PVPlayer SDK API document for that platform.

13.2 Error Codes

When PVPlayer engine reports an error, the error code would be one of PVMF status codes that provides a high-level description of the error. PVPlayer engine specific error code would be sent with the PVMF status code in the event extension interface pointer (PVInterface*) if available. The player engine specific error code would be encoded in the object pointed by the interface pointer and can be retrieved using PVMFErrorInfoMessageInterface extension interface methods.

PVMF status codes are defined in pvmf_return_codes.h. PVPlayer engine specific error code would be in the range from 1024 to 8191 as specified by PVPlayerErrorEventType enum in pv_player_interface.h. The UUID for PVPlayer engine specific error code collection and event codes are defined in pv_player_interface.h as PVPlayerErrorInfoEventTypesUUID.

The PVPlayer SDK uses the following PVMF status codes for error events.

- `PVMFErrCancelled`
- `PVMFErrNoMemory`
- `PVMFErrNotSupported`
- `PVMFErrArgument`
- `PVMFErrBusy`
- `PVMFErrNotReady`
- `PVMFErrCorrupt`
- `PVMFErrTimeout`
- `PVMFErrOverflow`
- `PVMFErrUnderflow`
- `PVMFErrInvalidState`
- `PVMFErrNoResources`
- `PVMFErrResourceConfiguration`
- `PVMFErrResource`