Apple, Inc. v. Motorola, Inc. et al Doc. 244 Att. 5

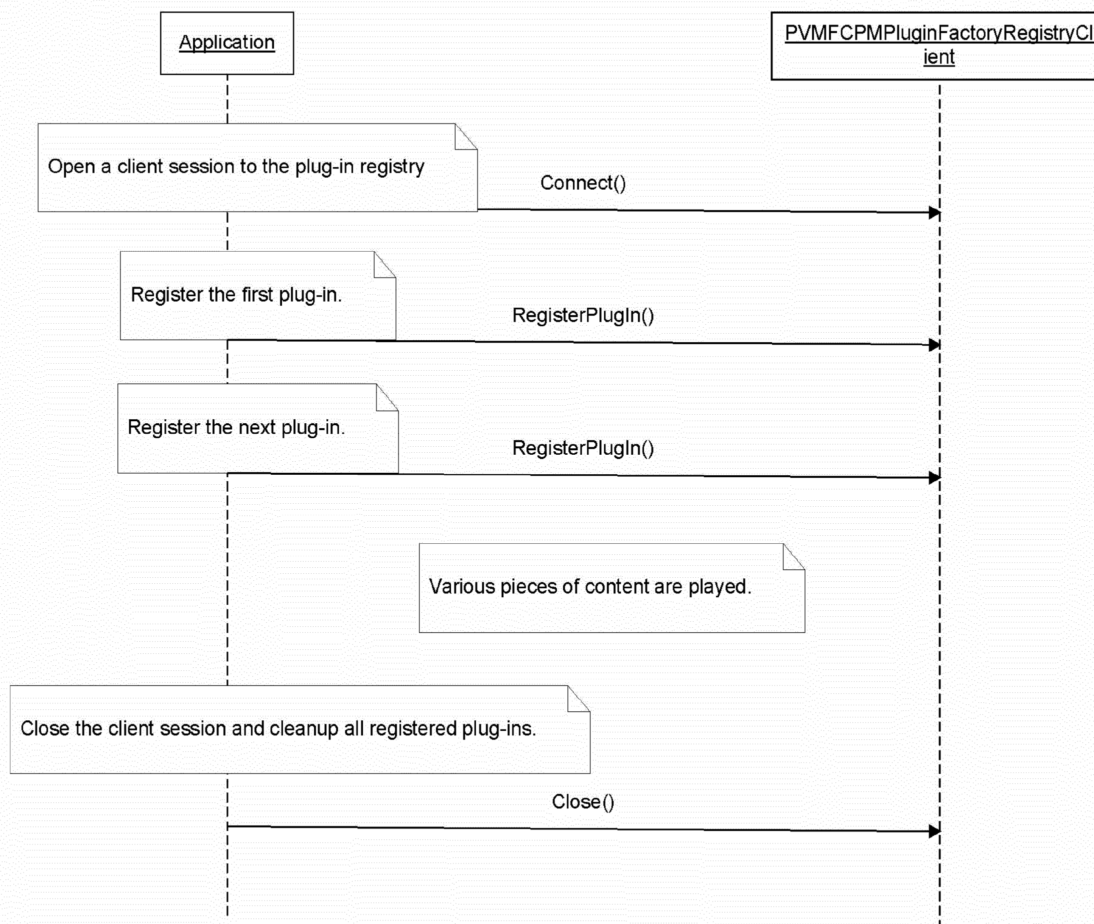

Figure 28: Preparation Sequence to Play DRM Protected Contents

14.10.2 Playback of DRM Content with a Valid License Available

Playback of DRM content with a license available is very similar to the usual playback sequence. The player engine will obtain the license from the license store during the Init phase and report success from the Init command if the license is valid. At that point the usual playback sequence can happen just as with non-protected content. If there is not license for this content in the license store or the existing license is invalid (e.g., expired), then the player engine will report an error and the sequence will proceed as described in Section 14.10.3. The sequence diagram below shows the details of the interaction between the application and player engine for the case of an available valid license.

WI-Apple1696005
Dockets.Justia.com

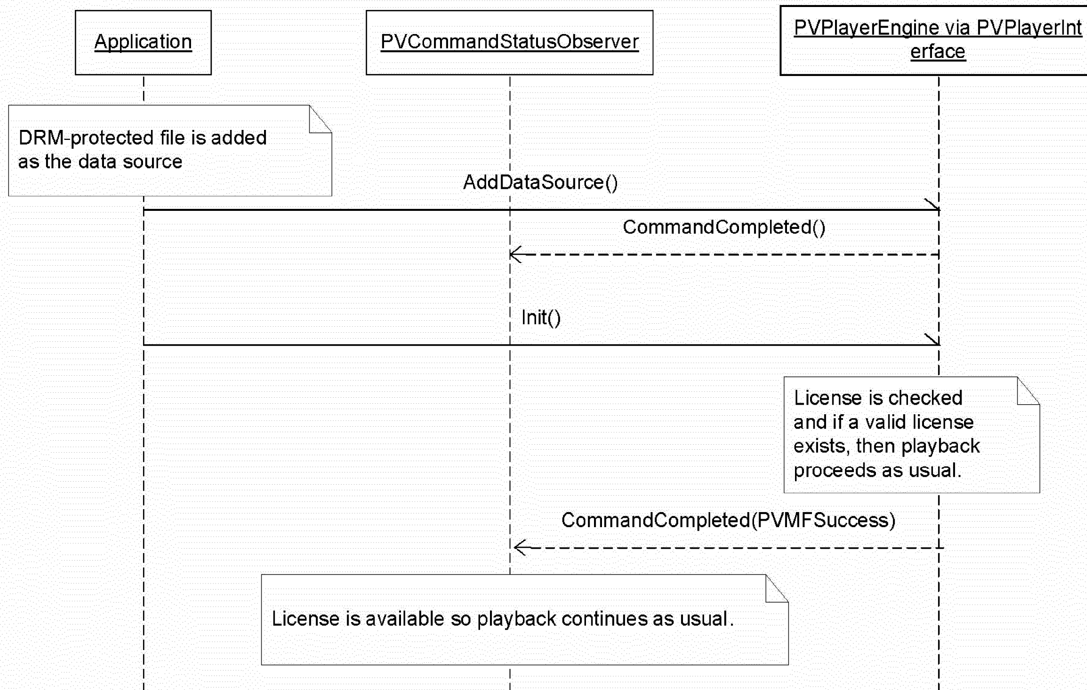

Figure 29: Playback of DRM Content with a Valid License Available

14.10.3 Playback of DRM Content *without* a Valid License Available

If there is no license at all for this content in the license store or the existing license is not valid, then the application will need to send the engine an explicit request to attempt to acquire the license. The call to Init will fail with the return code PVMFLicenseAcqRequired indicating that a license must be obtained before the clip can be played. At this point, the application can fail the playback and end the session with the player engine, acquire the license through some external process (i.e., outside the scope of the player engine), or request that the engine attempt to acquire the license.

In order to request that the engine acquire the license, the application must first get access to the appropriate extension interface of the player engine. This process is same as for any extension interface to engine, the QueryInterface is called with the relevant interface ID. The extension interface is returned in the CommandCompleted call to the observer. Once the application has the license acquisition interface, it can make the call to AcquireLicense. The AcquireLicense call takes some opaque data as a parameter. This data can be retrieved through the player meta-data interface using the “drm/dla-data” key. The engine attempts to get the license and returns the result in the CommandCompleted callback to the observer. Assuming the license acquisition was successful, the application can proceed with Init call again followed by the usual sequence for playback. Once the application has finished with the license acquisition interface, it should free the resource by calling the interface’s removeRef() method. The sequence is shown in the diagram below.

WI-Apple1696006

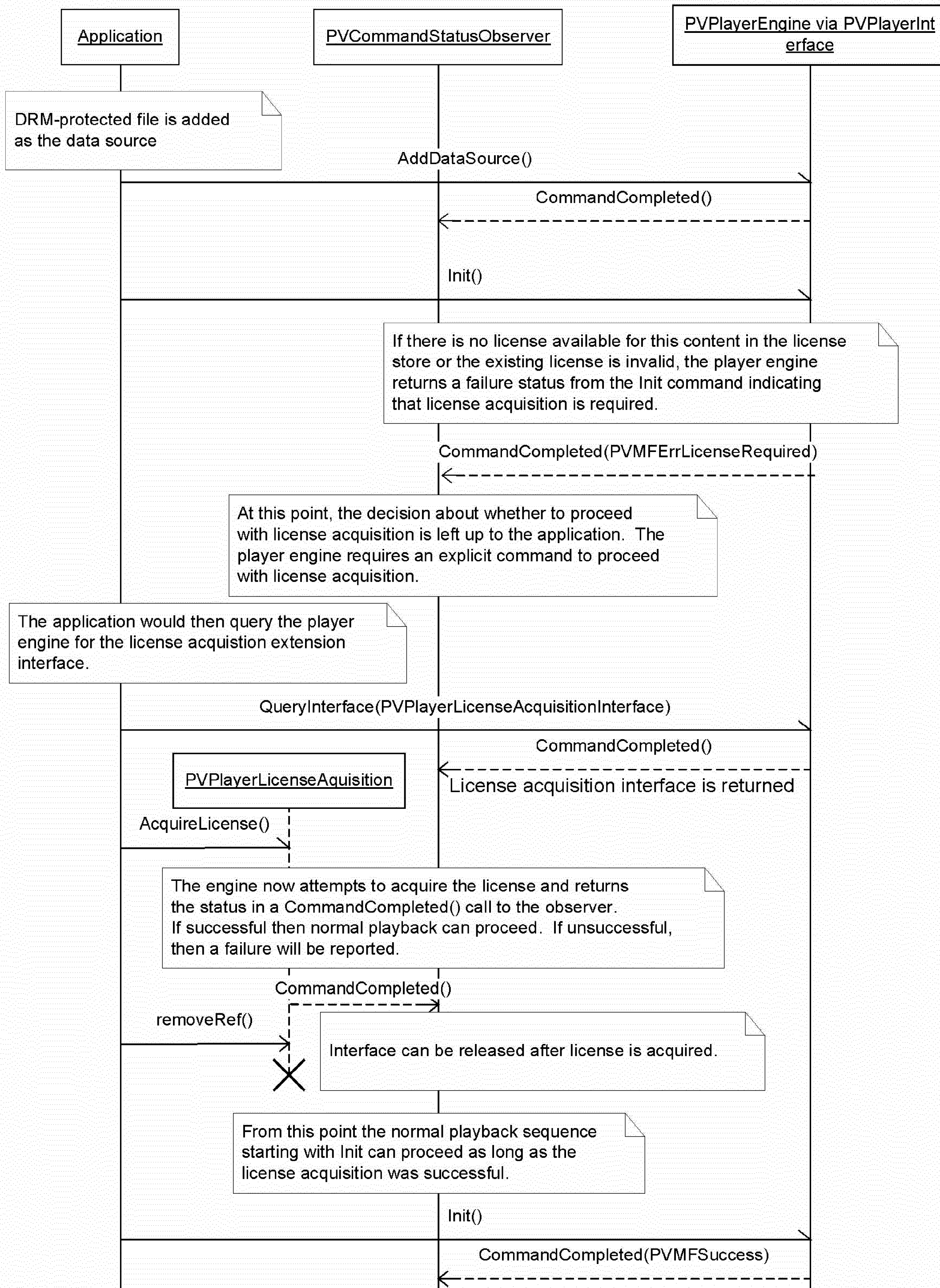

Figure 30: Playback of DRM Content without a Valid License Available

WI-Apple1696007

14.10.4 Cancel the License Acquisition of DRM Content

The AcquireLicense call can be canceled before the engine returns the result in the CommandCompleted callback to the observer. It can make the call to CancelAcquireLicense. Once CancelAcquireLicense is called during the license acquisition, the engine attempts to cancel the license acquisition and returns the result in the CommandCompleted callback to the observer. The call to AcquireLicense will be returned with the return code PVMFErrCancelled, if cancel was successful. The sequence is shown in the diagram below.

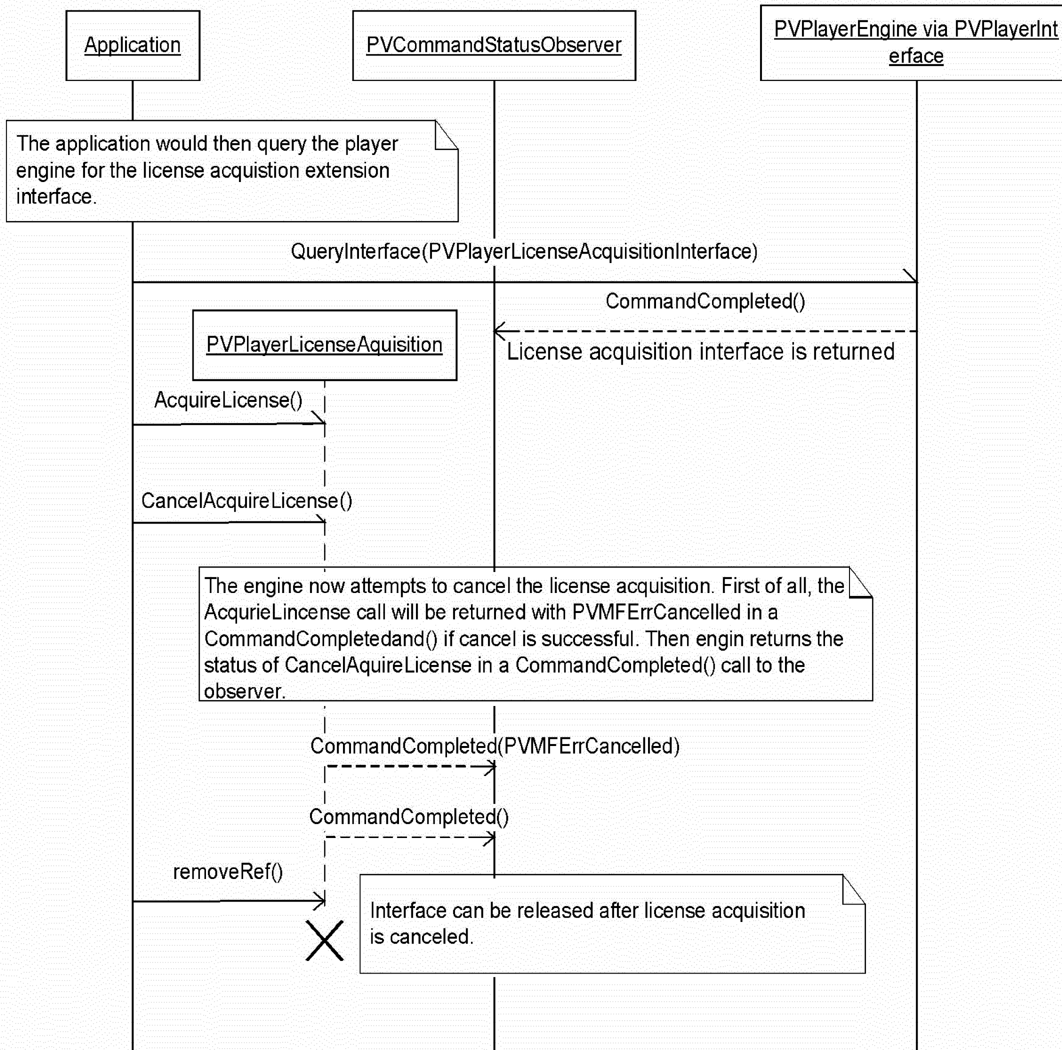

Figure 31: Cancel License Acquisition

WI-Apple1696008

14.10.5 Preview of DRM Content *without* a Valid License Available

A variation on the scenario covered in Section 14.10.3 is the case where there is no valid license for full playback of a piece of content, but there it can be previewed. This scenario might be a common way of initially distributing content so that consumers can preview it before deciding to purchase a full license. In this case the Init() method will return with the code

PVMFErrLicenseRequiredPreviewAvailable, which indicates that a license is required for full playback but a preview is available. In order to play the preview, the application must remove the current source then add it back with a flag set on the local data source to indicate preview mode. The sequence diagram below shows the interaction between the engine and the application.

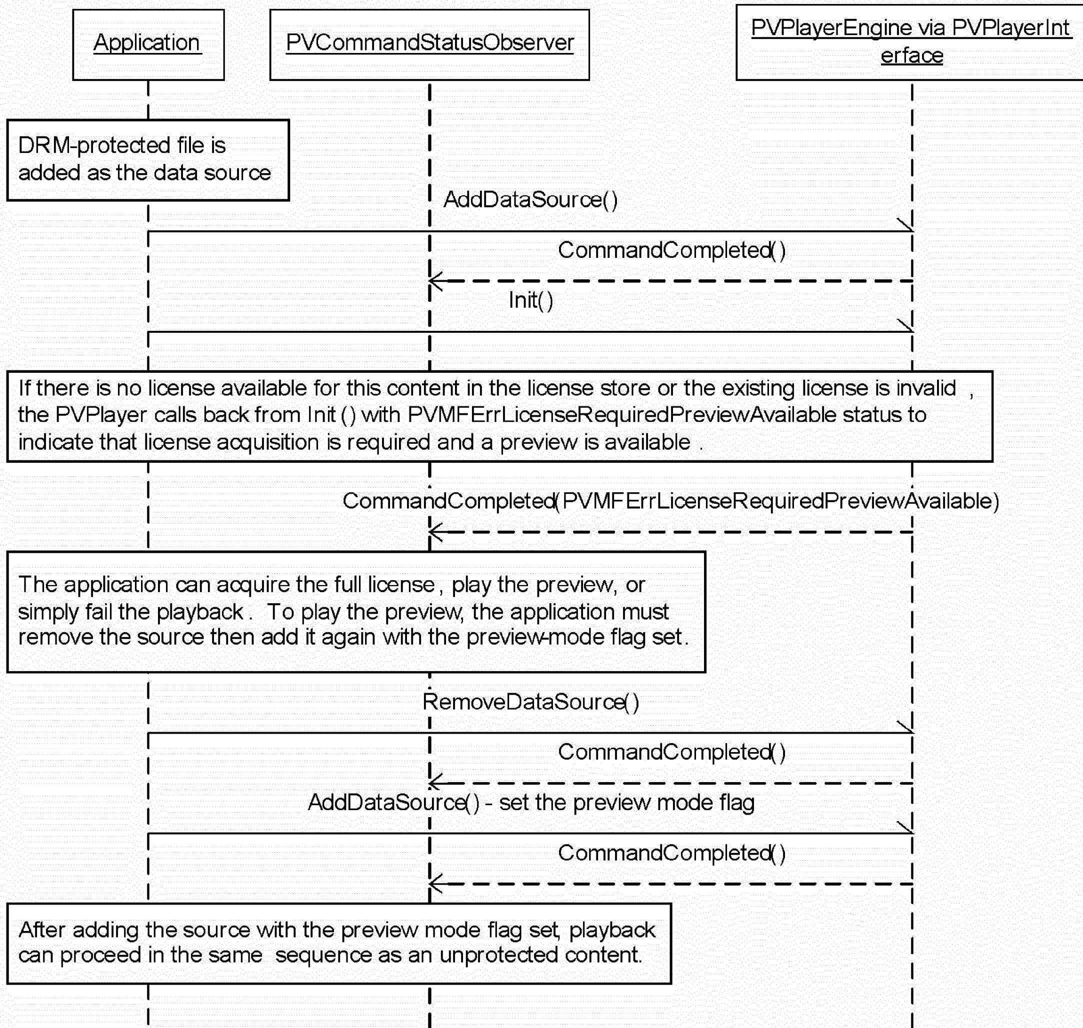

Figure 32: Preview of DRM Content without a Valid License Available

WI-Apple1696009

14.10.6 Playback of DRM Content with Auto License Acquisition

In cases where the license for the content exists in the license store, but it is invalid, the may attempt automatically acquire a valid license if it has been marked for auto-acquisition (handled outside of the player engine). In that case the engine will automatically attempt the license acquisition during the Init phase and return the status to the application in the CommandCompleted call to the observer. In the process the engine will send an informational event to the event observer as a notification that the license acquisition is happening. The sequence is shown in the diagram below.

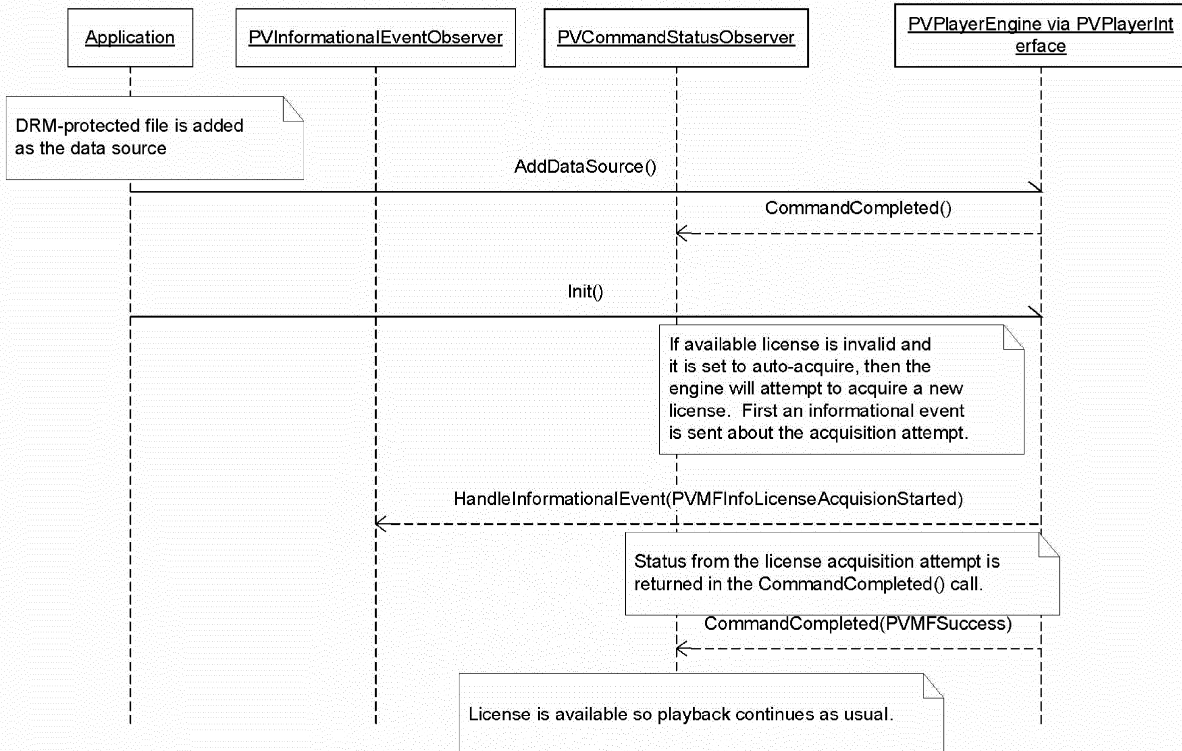

Figure 33: Playback of DRM Content with Auto-Acquisition of the License

WI-Apple1696010

14.11 Using SetPlaybackRange and PVMFInfoEndOfData Event

The SetPlaybackRange() method can be used to set the end time of playback at which point PVPlayer will send a PVMFInfoEndOfData event and stop playback. If no end time was specified, that event gets sent upon reaching the end of the stream instead. Information on how to obtain the complete duration can be found in section Metadata Handling.

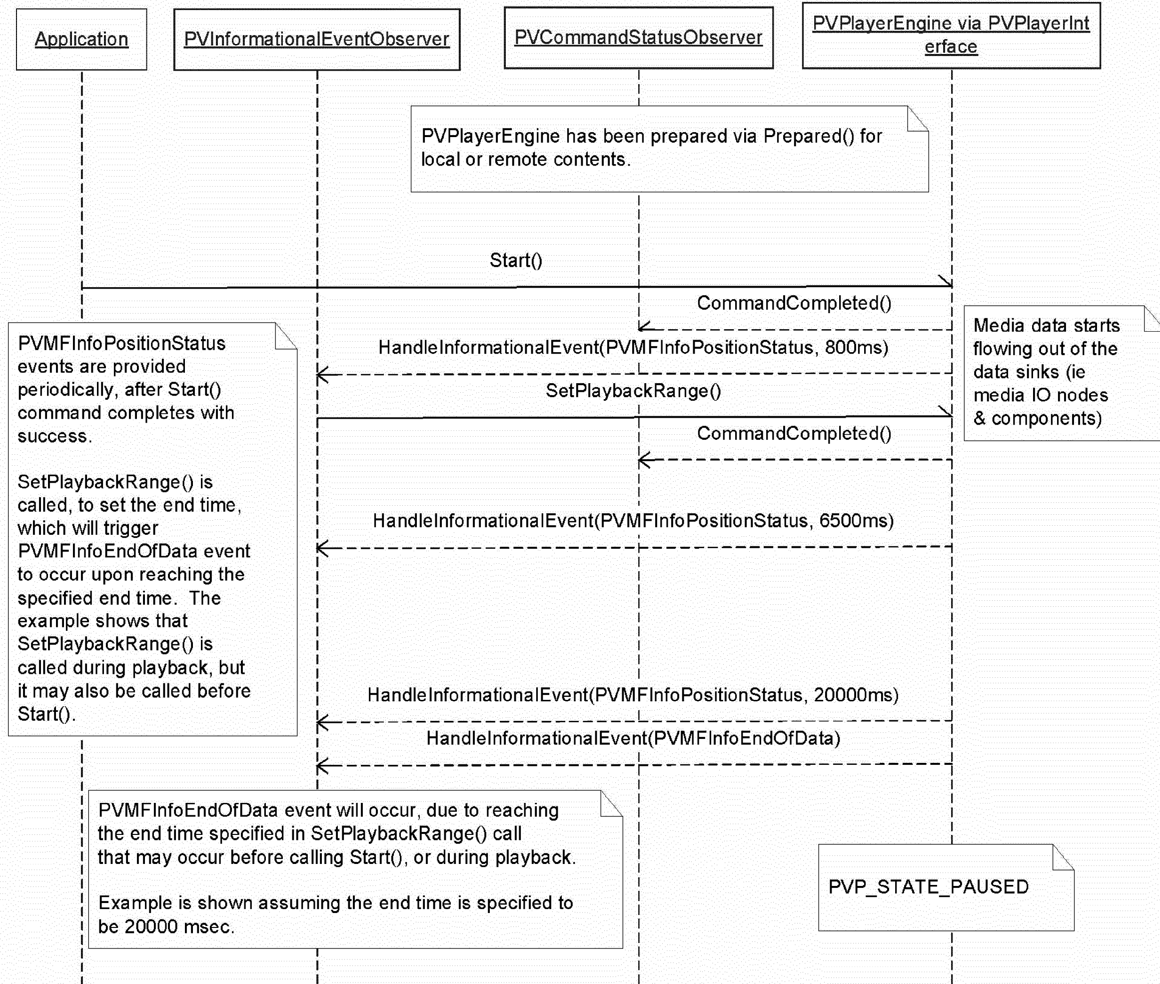

Figure 34: Using SetPlaybackRange and PVMFInfoEndOfData Event

WI-Apple1696011

14.12 Looped Playback Using SetPlaybackRange

Applications can achieve looped playback by repositioning the playback to the beginning of the file or stream using SetPlaybackRange() upon receiving the PVMFInfoEndOfData event.

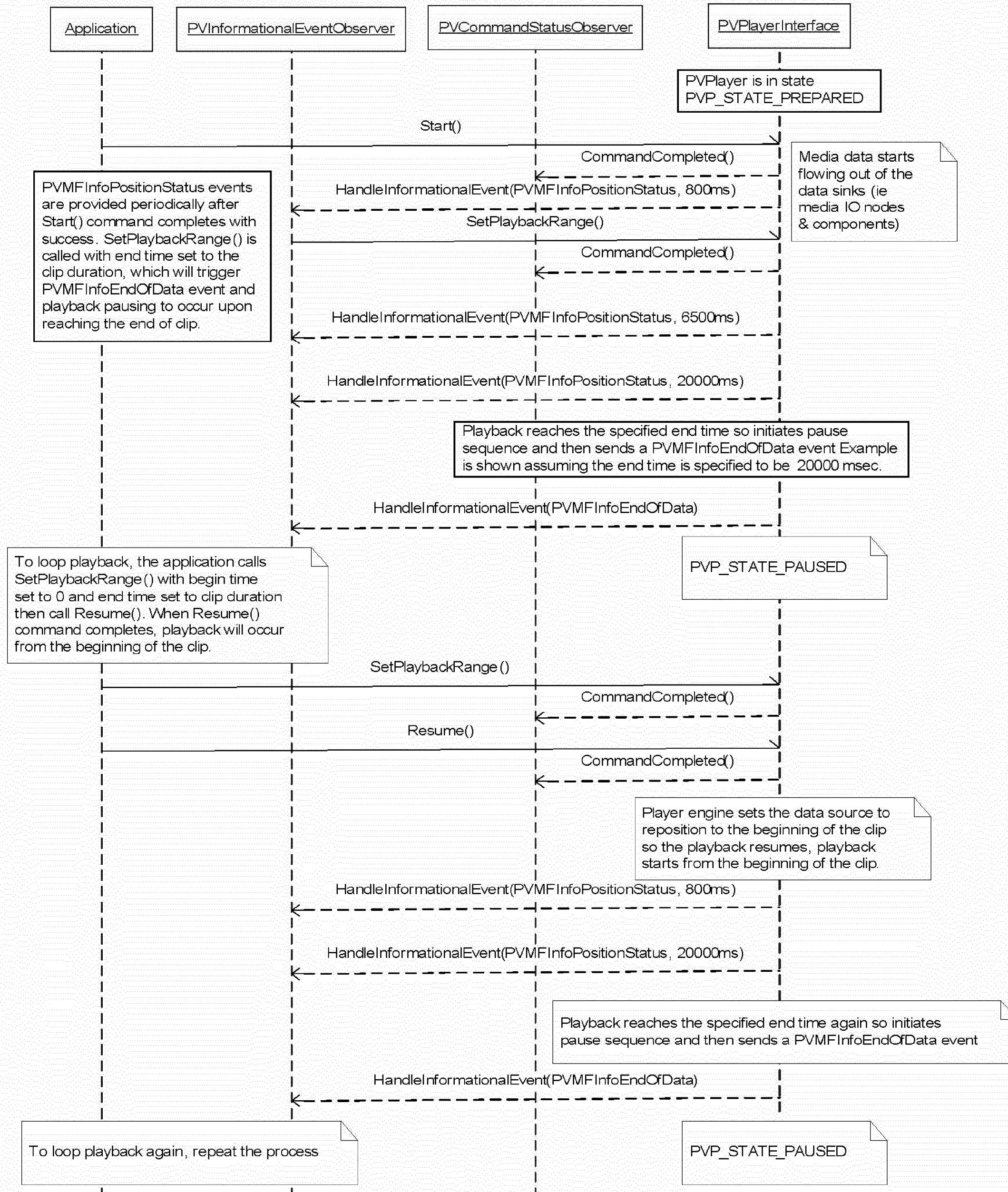

Figure 35: Looped Playback Using SetPlaybackRange

WI-Apple1696012

14.13 Start Download Session

To start a download and playback session, the application would need to configure the PVPlayerDataSourceURL object to provide the download source URL, playback control mode and other information. The following playback control modes are defined in PVMFSourceContextDataDownloadHTTP class:

1.ENoPlayback – Download only and playback will not be started
2.EAfterDownload – Playback will start after download is completed
3.EAsap – Progressive Download mode where playback starts as soon as possible.
4.ENoSaveToFile – Progressive Streaming mode where downloaded media data is not stored in file.

The sequence below illustrates the interaction between application and PVPlayer to start a download playback session.

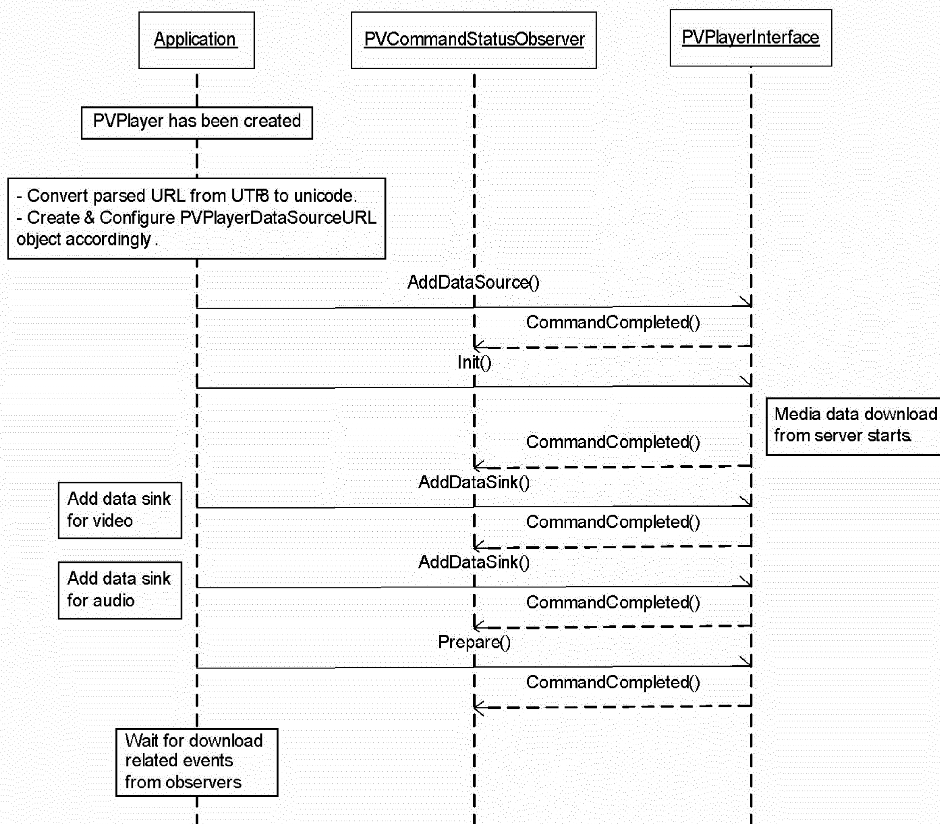

Figure 36: Start Download Session

WI-Apple1696013

14.14 Handling Progressive Download Events

This diagram shows a progressive download session that starts playback as soon as enough media data has been downloaded.

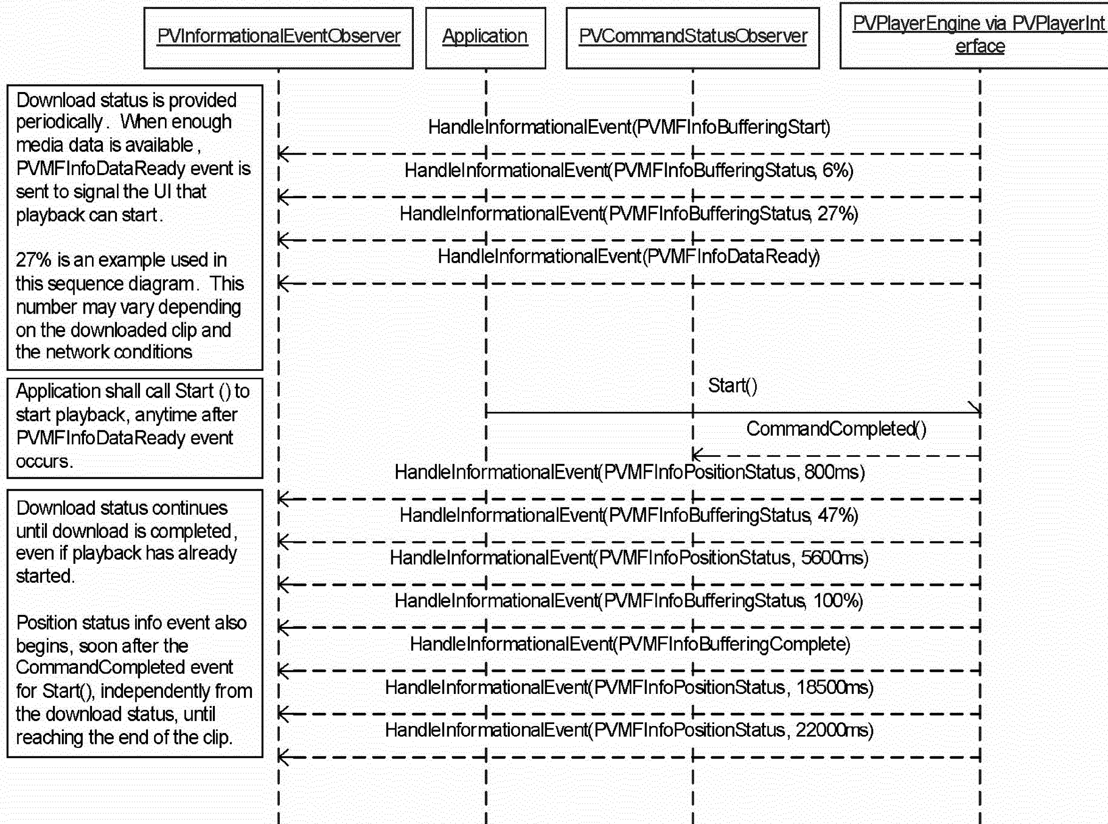

Figure 37: Handling Progressive Download Events

WI-Apple1696014

14.15 Handling Download Events

This diagram shows a download session that starts playback after the entire file is downloaded.

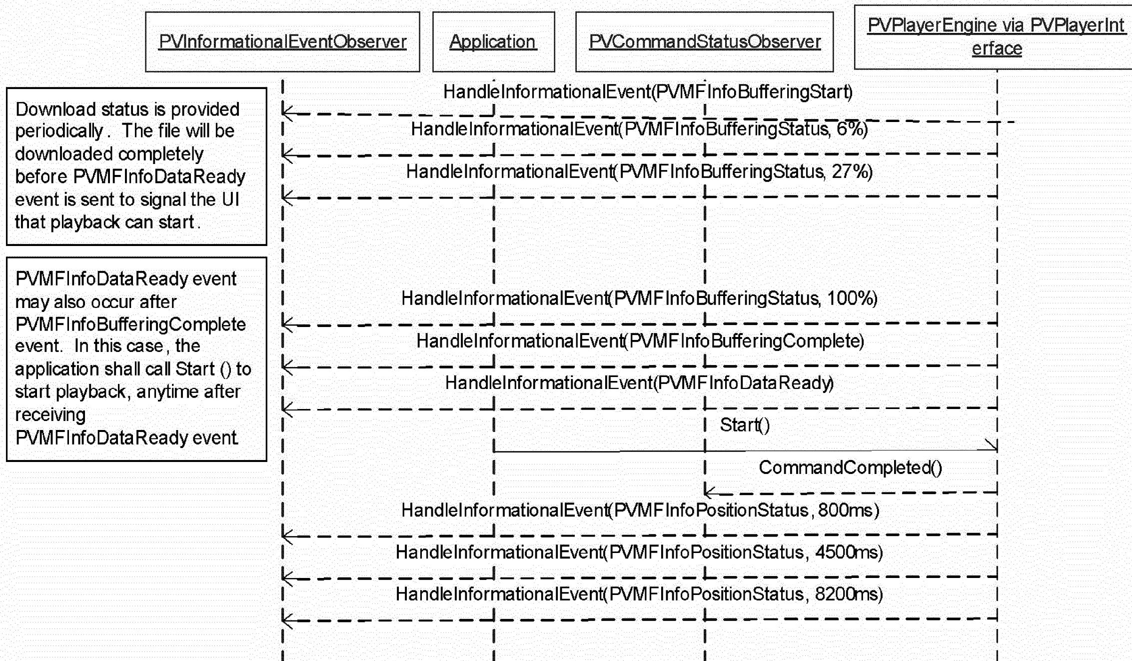

Figure 38: Handling Download Events

14.16 Auto-Pause-Resume in Progressive Download Session

In progressive download session, it is possible that the playback is consuming data from the partially downloaded media file faster than the download speed due to network condition and/or buffering condition settings. When PVPlayer runs out of data for playback, it would trigger the auto-pause sequence and notifies the application by sending PVMFInfoUnderflow event. When more data is downloaded during auto-pause, PVPlayer auto-resumes the playback and notifies the application by sending PVMFInfoDataReady event. It is not necessary for the application to call Start() again upon PVMFInfoDataReady event for auto-resume notification.

WI-Apple1696015

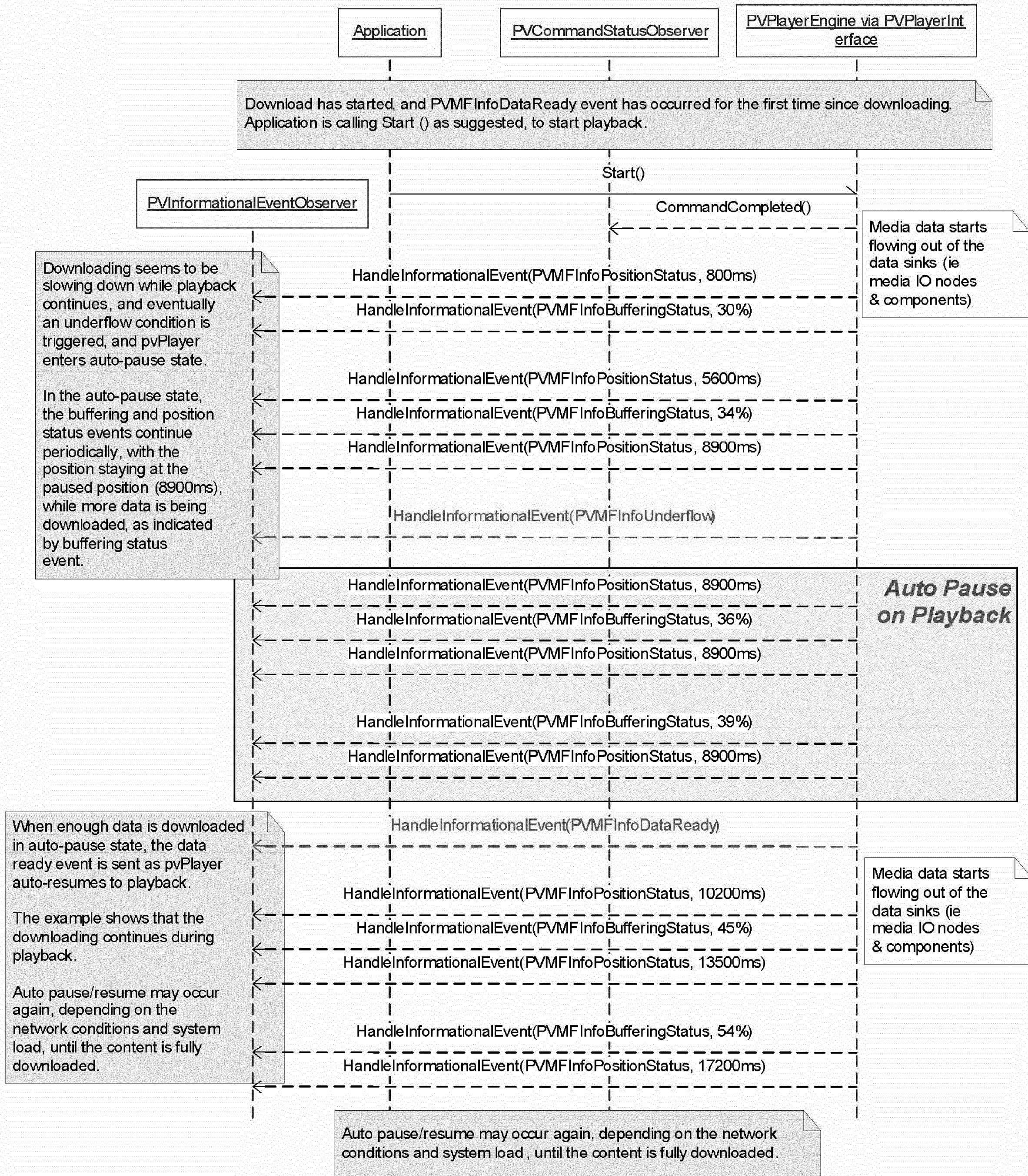

Figure 39: Auto-Pause-Resume in Progressive Download Session

WI-Apple1696016

14.17 Error Recovery During Initialization

When processing the Init() request, PVPlayer engine receives an error from the source node (e.g. file not present, network not available). PVPlayer engine goes to the ERROR state, handles the error by resetting the source node, and goes back to the IDLE state. When error handling is complete, PVPlayer engine reports PVMFInfoErrorHandlingComplete informational event.

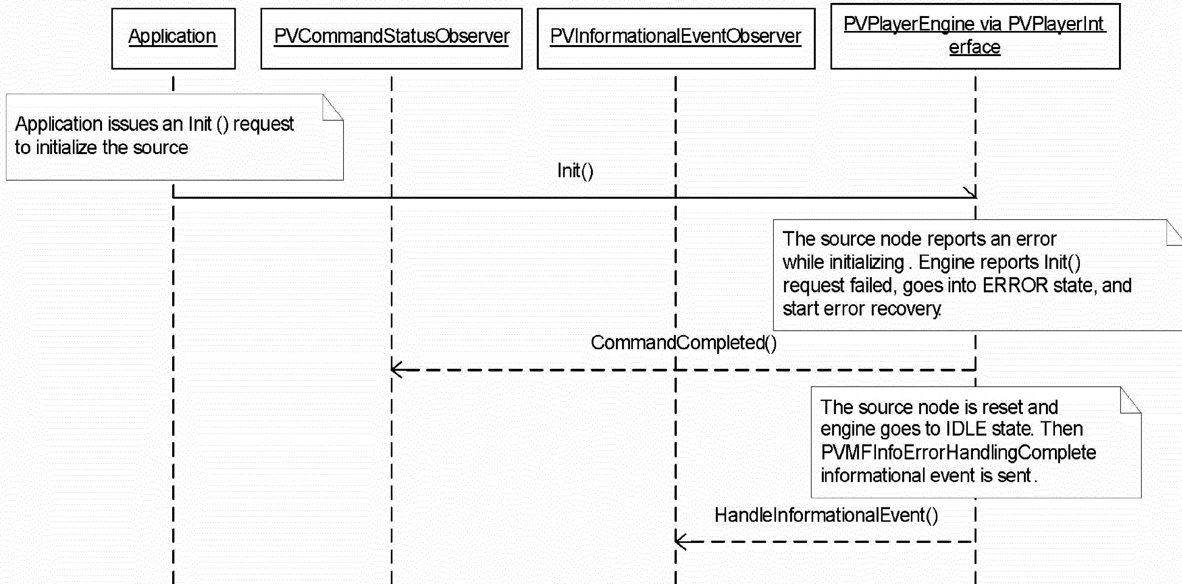

Figure 40: Error Recovery During Initialization

14.18 Error Recovery During Playback

During playback, PVPlayer engine receives an error from a decoder node (e.g. device became unavailable, corrupt data). PVPlayer engine goes to the ERROR state, handles the error by stopping the playback, and goes back to the INITIALIZED state. When error handling is complete, PVPlayer engine reports PVMFInfoErrorHandlingComplete informational event.

WI-Apple1696017

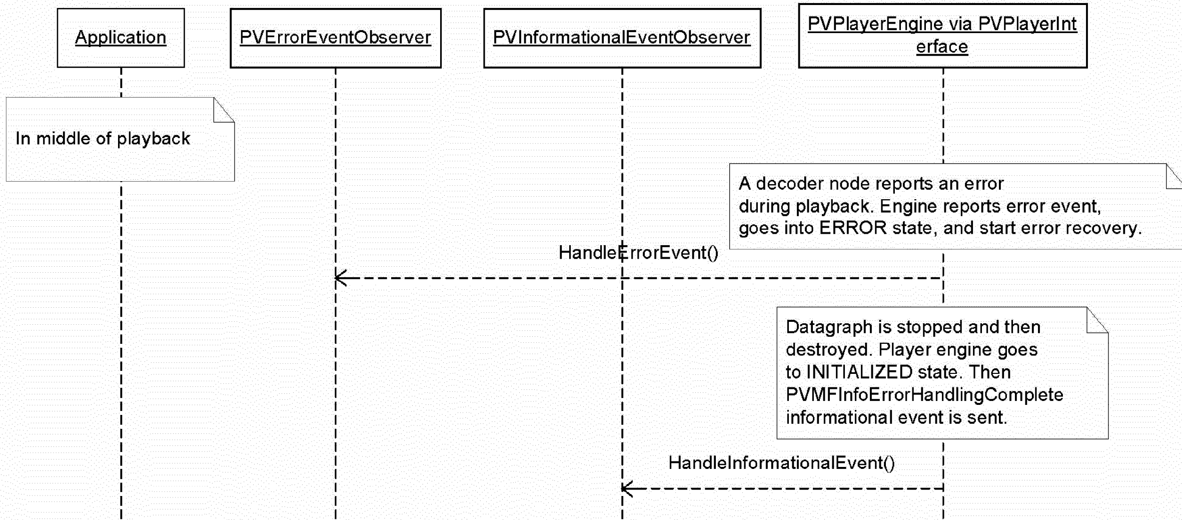

Figure 41: Error Recovery During Playback

14.19 Unrecoverable Error Handling

During playback, PVPlayer engine receives an error from the source node which requires the node to be destroyed. PVPlayer engine goes to the ERROR state, handles the error by stopping the playback, then resetting the source node, and cleaning up. The engine ends up in the IDLE state. When error handling is complete, PVPlayer engine reports PVMFInfoErrorHandlingComplete informational event.

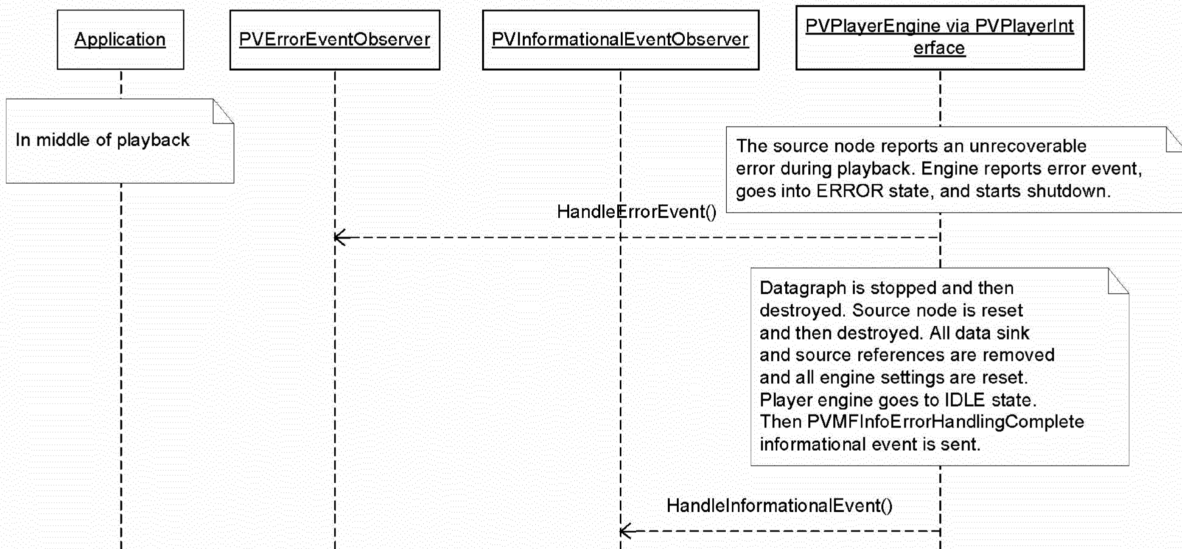

Figure 42: Unrecoverable Error Handling

WI-Apple1696018

15 Application's involvement in Track Selection

It is possible for the user of PVPlayerSDK (also referred to as application or app) to participate in the track selection process. The extension interface (PVPlayerTrackSelectionInterface) is exposed via the player API, QueryInterface(), by requesting with the UUID associated with the interface. Before describing this interface and its usage we define the following terms:

- Complete List – This is the complete list of available tracks in multimedia presentation.
- Playable List – This is the list of playable tracks. The list of playable tracks is a subset (at times a proper subset) of the complete list of available tracks.
- Selected List – This refers to the list of tracks selected by player engine from playable list for playback. Player engine performs track selection during "prepare".
- PVMFMediaPresentationInfo – Player engine uses this data structure to expose the contents of the above three lists.
- PVMFTrackSelectionHelper – This is an abstract interface that exposes a synchronous method to obtain track selection inputs from the user of PVPlayerSDK. If the user of PVPlayerSDK wishes to participate in the track selection process then, the user of the SDK needs to provide an implementation of this object. If provided, PVPlayerSDK will invoke the SelectTracks(...) API as part of "prepare".

PVPlayerTrackSelectionInterface exposes APIs to retrieve the complete list, playable list and selected list. In addition to these APIs this interface also provides a mechanism for the application to register its implementation of the PVMFTrackSelectionHelper interface. While invoking the "SelectTracks(…)" API, PVPlayerSDK provides it with a playable list and the implementation of PVMFTrackSelectionHelper is responsible for creating the selected list based on this input. For exact syntax of these interfaces and their APIs please refer to \engines\player\include\pv_player_track_selection_interface.h

15.1 Memory Considerations

All of the APIs of PVPlayerTrackSelectionInterface and PVMFTrackSelectionHelper objects allocate memory. Therefore both these interfaces contain explicit release APIs. These release APIs remove any ambiguity about memory ownership.

WI-Apple1696019

16 Diagnostics

16.1 Instrumentation and Debug Logs

The PVLogger component is used inside PVPlayer SDK to log stack trace, warnings, error, and other information. PVLogger provides a very flexible and extensible framework that allows fine-grained control of the exact logging point and logging level along with the ability to filter messages and route messages to arbitrary outputs.

OSCL-based PVPlayer engine expects PVLogger to be initialized by the user of PVPlayer SDK and it provides standard SDK APIs for logging via the extension interface. Refer to the PVLogger User's Guide for more details on PVLogger.

WI-Apple1696020