PVAuthor Developer's Guide
OHA 1.0, rev. 1
Sep 8, 2009

Table of Contents

List of Figures

1. Introduction

This document is a guide for developers using the PVAuthor engine APIs. A client of PVAuthor engine can be an application or an adapter layer used to map the PVAuthor engine interface to a different framework or API layer used by the application. This documents describes how to use PVAuthor engine interface and its extensions to create, configure and control a multimedia authoring session.

The PVAuthor engine is the part of the PacketVideo Multimedia Framework (PVMF) that provides media recording capabilities for its clients. It is capable of capturing audio, video, and text media data, encoding media data to compressed formats, and multiplexing the encoded media data to various file formats. The input media data is typically provided by live source(s) such as camera and microphone, or in other cases, provided by unencoded media data files. The input data is then encoded to the data formats selected by the client, followed by multiplexing of media data into the selected file format. The multiplexed media data is written to the data sink provided by the client.

2. Architectural Overview

2.1. PVAuthor Structure

A client controlling the PVAuthor engine interacts through the PVAutthorEngineInterface and must also implement the PVAuthor engine observer interfaces in order to receive command completion, status information, and error information. The interface requires that media data sources and sinks are provided by the client for a recording session. The number and types of sources and sinks may vary depending on the properties of the recording session. These media sources and sinks should implement PVMFNodeInterface to allow PVAuthor engine to control them in a uniform way. Client control of the recording session is performed through PVAuthorEngineInterface. Figure 1 below shows the relationship between PVAuthor engine, the client, and other objects owned by the client. Figure 2 below shows a suggested list of attributes and operations of the client class. Note that the attributes listed here will be referred to in sequence diagrams later in this document.

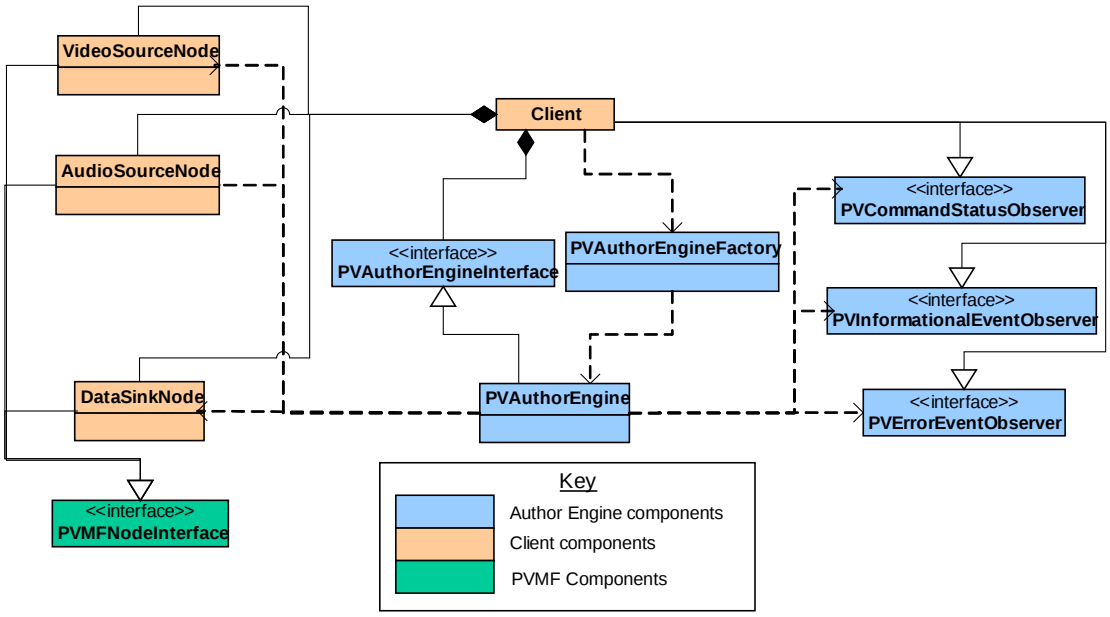

Figure 1: Class diagram of PVAuthor

Client
-iAuthor : PVAuthorEngineInterface* -iAudioSrcNode : PVMFNodeInterface* -iVideoSrcNode : PVMFNodeInterface* -iTextSrcNode : PVMFNodeInterface* -iSelectedComposer : void* -iComposerConfig : PVInterface* -iAudioEncoderConfig : PVInterface* -iVideoEncoderConfig : PVInterface* -iTextConfig : PVInterface*
+CommandCompleted(in aResponse : PVCmdResponse) +HandleInformationalEvent(in aEvent : PVAsyncInformationalEvent) +HandleErrorEvent(in aEvent : PVErrorEventObserver)

Figure 2: Attributes and operations of the client.

2.2. Overall Sequence Diagram

The diagrams in this section present the overall sequence of API calls to set up and control a recording session. Detail usage of the APIs will follow in subsequent sections in this document.

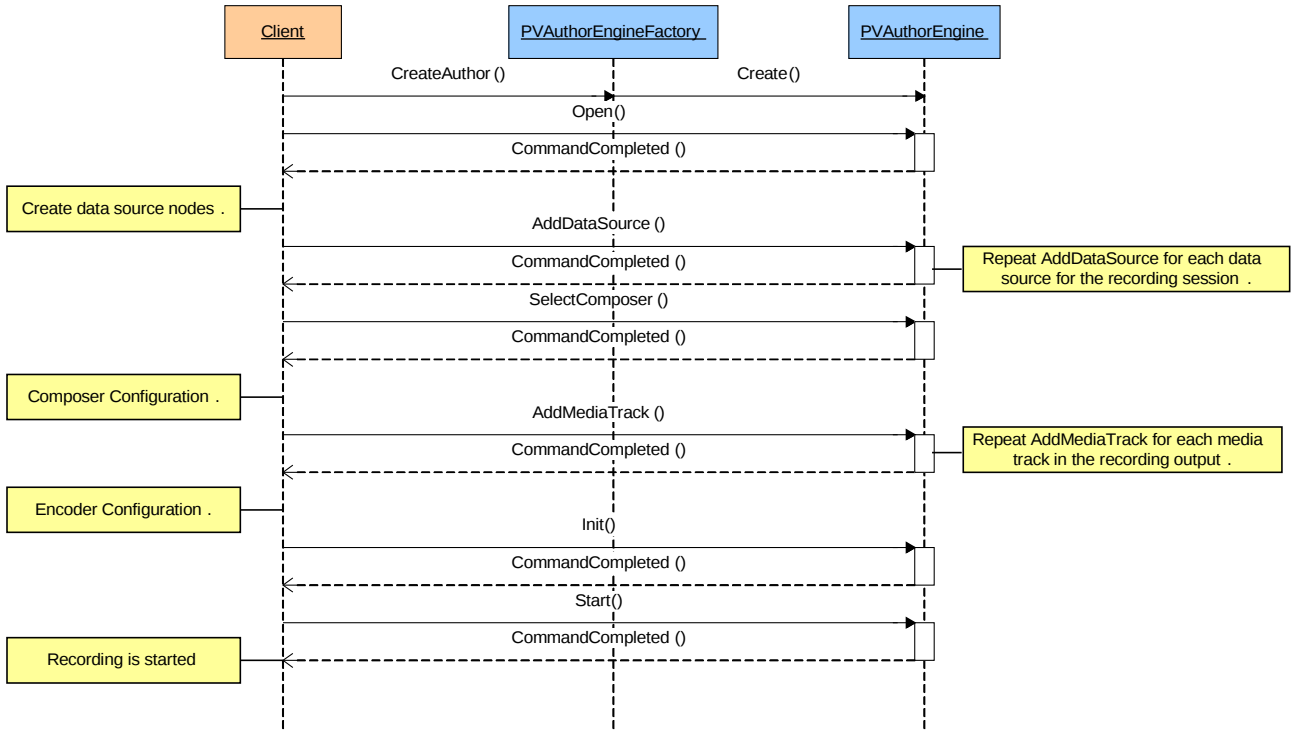

Figure 3: Overall Sequence Diagram

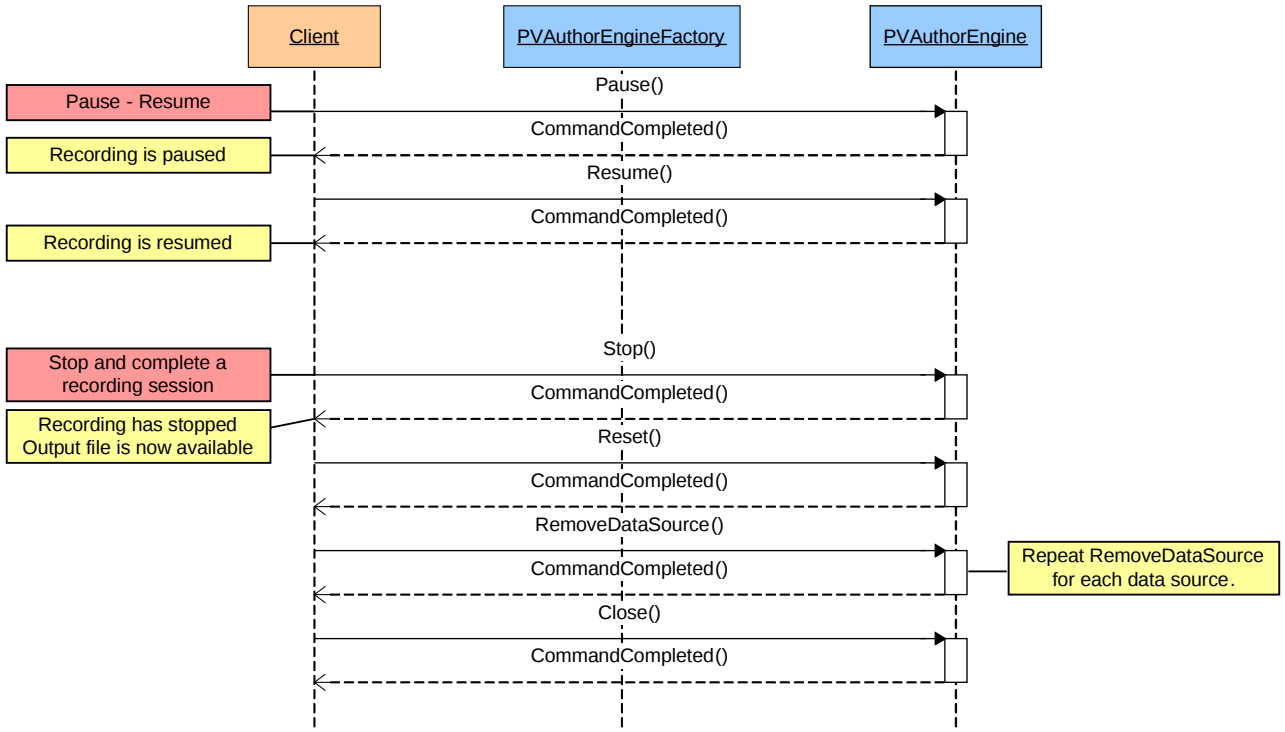

Figure 4: Overall Sequence Diagram Continued

3. PVAuthor State Machine

The PVAuthor engine has 6 states: Idle, Opened, Initialized, Recording, Paused, and Error. To transition from one state to another, the user will need to call the session control APIs of PVAuthorEngineInterface. Figure 5 shows the state transition diagram.

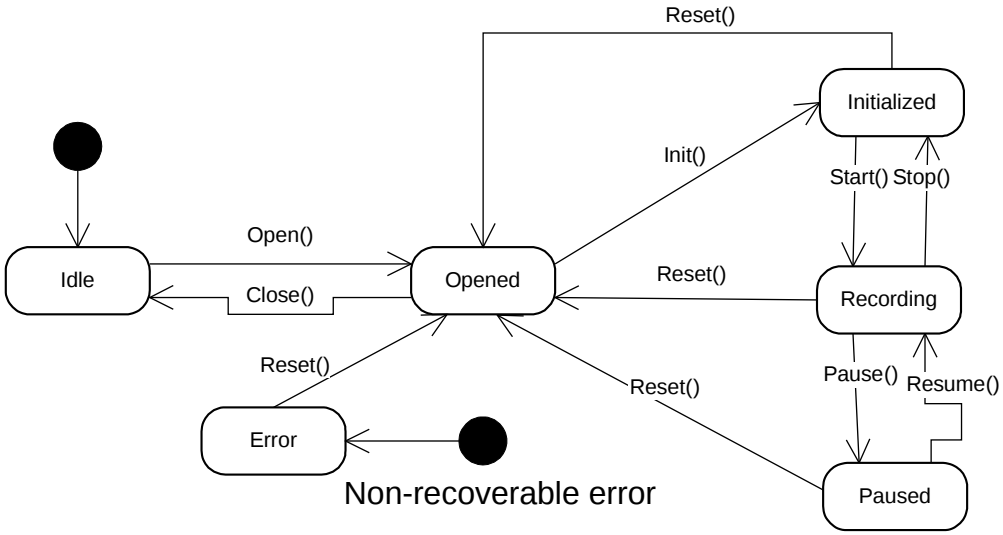

Figure 5: PVAuthor state transition diagram.

4. Create and Open Session

To create a recording session, the client needs to first create a PVAuthor engine object. This step is done through the Create() method in PVAuthorEngineFactory class, which will create a PVAuthor engine without an active recording session. To open a session, the client needs to call the Open() method on the PVAuthor engine object. Figure 6 illustrates the sequence of method calls to create and open a recording session.

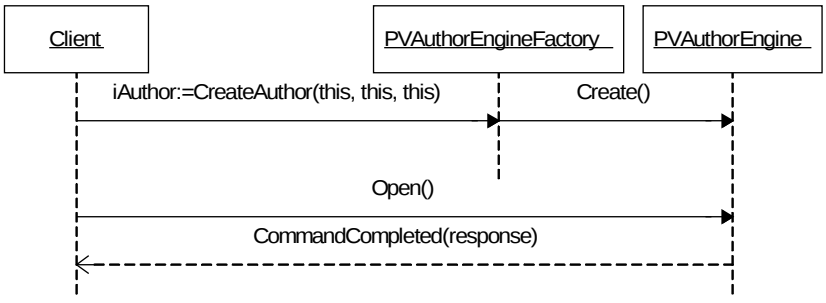

Figure 6: Create and open a recording session.

5. Data Sources

5.1. Create and Add Data Sources

As mentioned in the previous section, a client to PVAuthor engine needs to create media data source objects and provide them through the AddDataSource() method for capturing source data during the authoring session. The media data source objects are PVMF Nodes that wrap around the underlying drivers for capturing audio, video and text source data. A common method to integrate data sources to PVAuthor engine is using the media I/O interface and PvmfMediaInputNode. Please refer to the Media I/O Developer's Guide for information regarding the media I/O interface. Figure 7 illustrates the sequence to create media data sources and provide them to PVAuthor engine.

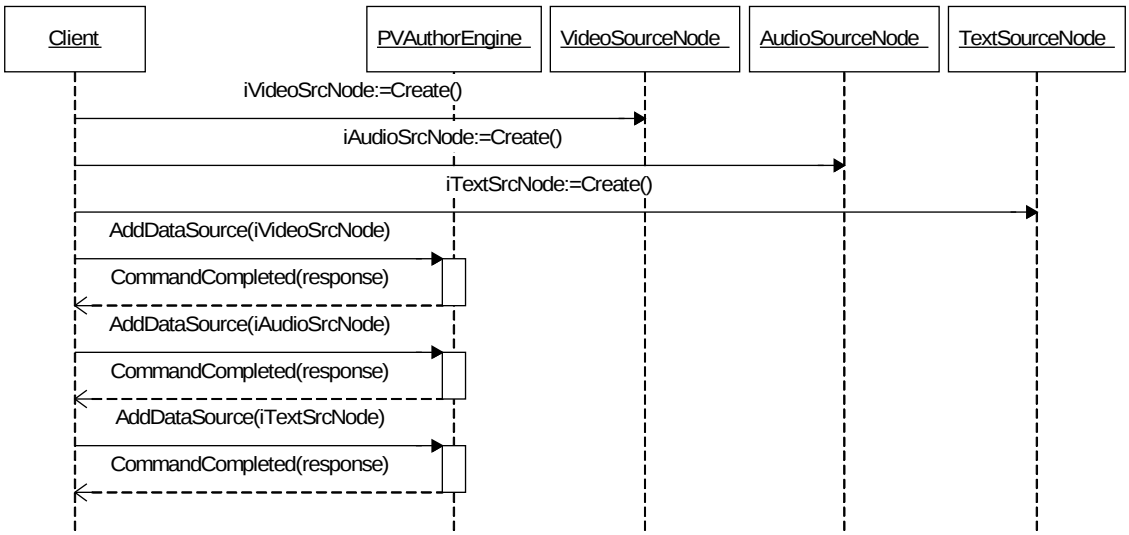

Figure 7: Create media sources and add to PVAuthor engine

5.2. Data Source Configuration

Besides creating the data source object and adding them to PVAuthor engine, the client is also responsible for configuring the data sources to capture source data in the desired format or properties. The available options for configuration are dependent on the data source node implementation and the capability of the underlying capturing devices. Please refer to the documentation of the data source nodes and capturing devices for the options available.

6. File Format Composer

The next step in setting up a recording session is to select a file format composer for the session. Multiplexing encoded media data and formatting the multiplexed data into the desired file format is functionality provided by the PVMF framework made available to the client through PVAuthor engine. The client is responsible for selecting a composer type, and configuring the composer through a configuration object provided by PVAuthor engine.

6.1. Composer Selection

Composer selection is done by calling SelectComposer() method on the PVAuthor engine object. The client would specify the Mime type of the composer to use, and a pointer to hold the configuration object PVAuthor engine provides. Alternately, the client can specify the Uuid of the composer to be used instead of the Mime type if such information is available to the client. When the method completes asynchronously, an opaque identifier for the selected composer is returned to the client in the response data. The client needs to store this opaque identifier and use it to identify the selected composer in PVAuthor engine API calls when necessary. Figure 8 below illustrates the sequence of method calls to select a composer.

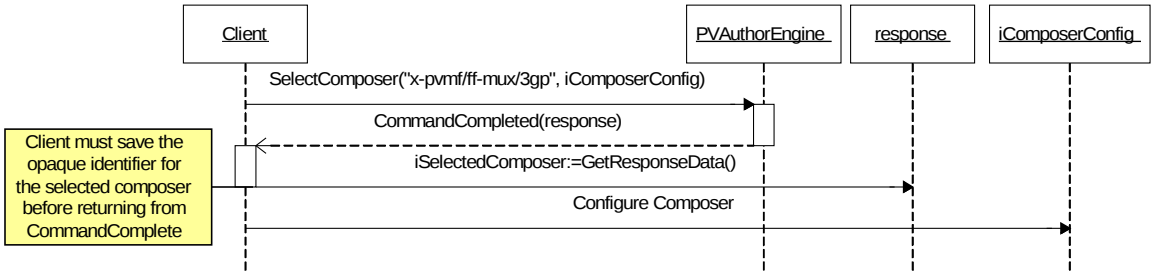

Figure 8: Composer selection

6.2. Composer Configuration

Composer configuration is done through the composer configuration object returned by author engine from the SelectComposer method call. The configuration interface implemented by this object varies depending on the composer selected. The client can check for the interface supported by the configuration object by calling the queryInterface method on the configuration object. The client can call the configuration methods after SelectComposer is completed unless specified otherwise by the configuration interface.

6.2.1. 3GPP Composer

If the client selected 3GPP file format composer, the returned configuration object implements the PVMp4FFCNClipConfigInterface interface. The client is required to call the SetOutputFileName method to set the output file name for the authoring session. Furthermore, if the client wants to author a I-Motion file, then SetAuthoringMode method call is also required. Please refer to author

interface documentation for information on various authoring modes. Below is a sequence diagram to illustrate the sequence of calls to configure the settings mentioned above.

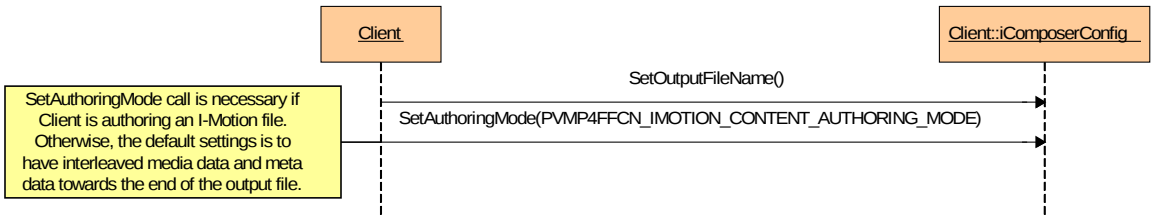

Figure 9: File name and authoring mode configuration

pvAuthorSDK authors files that are compliant to following specifications:

1. ISO Base Media File Format (ISO/IEC 14496-12 - [1])
2. MPEG-4 File Format (ISO/IEC 14496-14)
3. AVC File Format (ISO/IEC 14496-15)
4. 3GPP file format (3GPP TS 26.244)

pvAuthorSDK authors brands in "ftyp" box as per criteria listed below:

- If authored file contains ANY of the following media tracks
  - AMR
  - AMR-WB
  - MPEG4 Audio
  - MPEG4 Video
  - H263
  - AVC
  - TimedText

then pvAuthorSDK uses *3gp4* as major brand and *3gp4*, *3gp5*, and *mp41* as compatible brands.

- If authored file does NOT contain ANY of the media tracks listed below:
  - AMR
  - AMR-WB
  - H263
  - AVC
  - TimedText

then pvAuthorSDK uses *mp41* as major brand and *3gp4*, *3gp5* and *mp41* as compatible brands.

- If authored file does NOT contain ANY of the media tracks listed below:
  - MPEG4 Audio
  - MPEG4 Video

then pvAuthorSDK does not use *mp41* anywhere.

- If authored file contains ANY of the following media tracks
  - AMR
  - AMR-WB
  - MPEG4 Audio
  - MPEG4 Video
  - H263
  - AVC
  - TimedText

AND if it contains movie fragments then pvAuthorSDK uses *3gp6* as major brand and *3gp5*, *3gp6*, and *mp41* as compatible brands.

- pvAuthorSDK always adds *isom* as a compatible brand.
- pvAuthorSDK adds *mp42* as a compatible brand whenever it adds *mp41* as compatible brand (ISO/IEC 14496-14 does not require IODS to be present always. So we are compatible to this spec even though we are not authoring IODS box).
- pvAuthorSDK adds *avc1* as a compatible brand in case the authored clip has a H264 track.

The PVMp4FFCNClipConfigInterface also allows the Client to add optional meta data information to the output file. Below is a sequence diagram to illustrate the sequence of calls to add meta data. Please note that adding meta data information is optional and does not affect the validity of the output file whether the data is added or not.

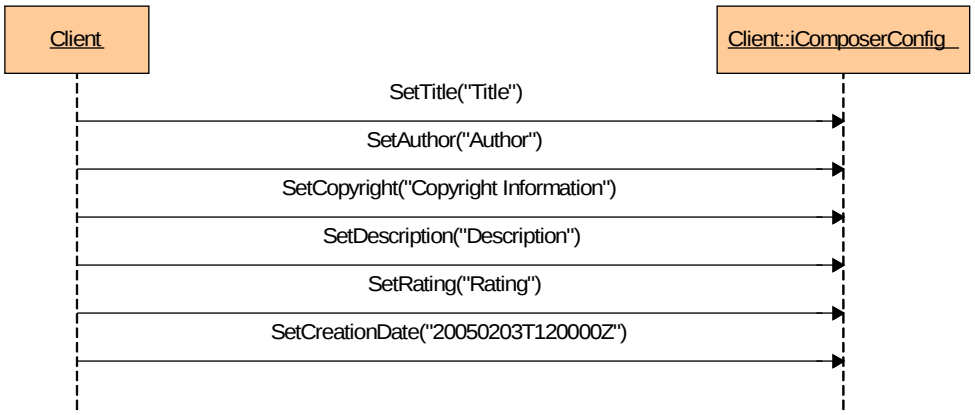

Figure 10: Adding Meta Data strings

The following metadata are currently supported through the PVMp4FFCNClipConfigInterface.

a. Title
b. Author
c. Copyright
d. Description

e. Rating
f. Creation Date
g. Artist or Performer
h. Genre
i. Classification
j. Key Words
k. Location Info.

6.2.2. AMR and AAC composer

When the client selects an AMR or AAC file format composer, the configuration object should implement the PvmfFileOutputNodeConfigInterface interface. The client is required to call the SetOutputFileName method to set the output file name for the authoring session. Please refer to Figure 9 for a sequence diagram.

7. Media Tracks

A media file, regardless of its format, should have at least one media track. Therefore, the client needs to add at least one media track to the file format composer in order to compose a valid output file. The maximum number of and types of media tracks supported varies depending on the properties of the selected file format composer. To create a file with multiple media tracks, for example AMR audio with H263 video and text tracks, the client will need to call AddMediaTrack for each of the track.

7.1. Add a Media Track

Adding of media track is done through the AddMediaTrack method. The client will need to specify the input PVMF node that provides the source data for this media track, the MIME type of the encoder to be used to encode the source data, and the file format composer in which a media track is added. The file format composer is identified by the opaque data returned in the CommandCompleted callback for the SelectComposer call. Alternately, the client can specify the Uuid of the encoder to be used instead of the Mime type if such information is available to the client. The client also needs to specify a PVInterface pointer where PVAuthor engine will save a pointer an instance of the configuration object of the selected encoder. Figure 11 below illustrates the sequence of calls to add a media track.

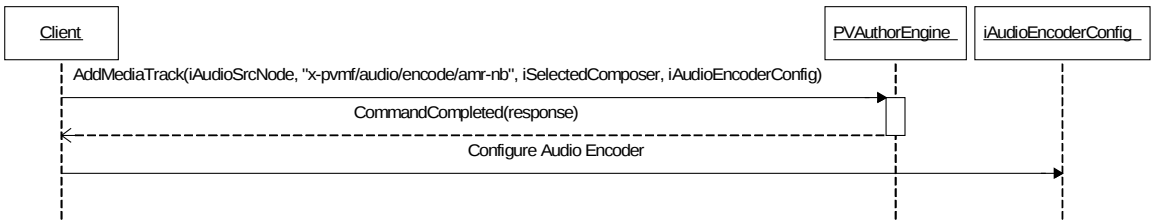

Figure 11: Add media track

7.2. Encoder Configuration

Encoder configuration is performed through the configuration interface object returned by the PVAuthor engine in the AddMediaTrack call. The configuration interface implemented by this object varies depending on the encoder selected. If the selected encoder supports no configuration interface, this pointer would be set to NULL. Also, if the input node provides encoded source data, no encoder would be selected for the media track by PVAuthor engine, and the configuration pointer would be set to NULL. The client can check for the interface supported by the configuration object by calling the queryInterface method on the configuration object. The client can call the configuration methods after AddMediaTrack call is completed unless specified otherwise by the configuration interface.

When the client selects an H263, MPEG4 or AVC video encoder, the configuration object should implement the PVMp4H263EncExtensionInterface interface. Please refer to PVAuthor engine interface documentation for the options available from the interface.

8. Data Sinks

Currently, PVAuthor only supports PV's 3GPP, AMR and AAC file format composer nodes. These nodes have integrated file IO and does not require the client to add data sink nodes for PVAuthor engine to write its output data to. The client will need to set the output file name through the configuration interface of the selected composer.

In future, when file format nodes without integrated file IO are supported, the client will be required to call AddDataSink method to specify the data sink node where a particular file format composer should write its output.

9. Additional Features Through Extension Interface

Besides features and configurations available from PVAuthor engine Interface and configuration objects provided by the Author engine, additional features are also available on demand through QueryInterface method. This methods allow PVAuthor engine to extend and expose new features to the client as they become available. Providing the client with UUIDs of features, enables the client to call QueryInterface to retrieve an interface object to use the feature. Figure 12 below shows the sequence of calls to query for an extension interface.

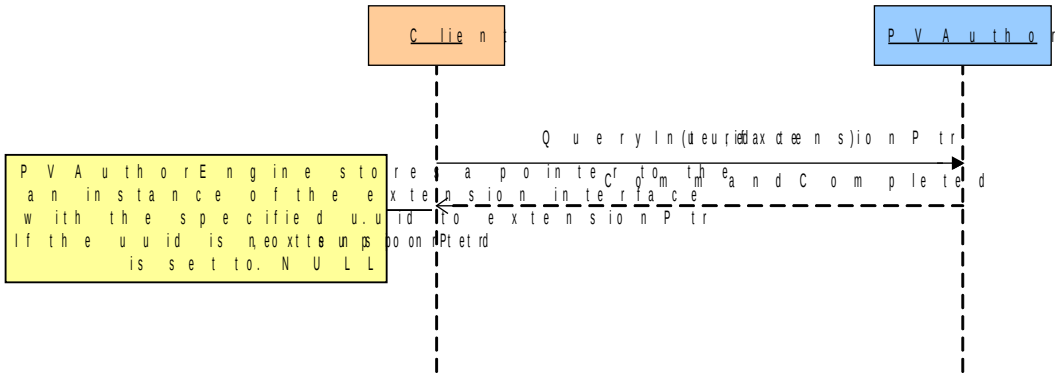

Figure 12: Query for extension interface

9.1. Max File Size, Duration and Progress Report

PVAuthor engine has a feature to set the maximum size of the output file or the maximum time duration of the output file. If the maximum file size or duration is set, PVAuthor engine will check against these settings while authoring the output file and stop the authoring session automatically when it reaches the specified maximum size or duration. Also, PVAuthor engine can be configured to provide progress reports to the client periodically. These progress reports can be in terms of file size written or the current duration of the output file. These features are available through the PvmfComposerSizeAndDurationInterface extension interface. To access these features, the client needs to first query for an instance of the extension interface, and then configure PVAuthor engine through the provided interface. Figure 13 shows the sequence diagram for these features.

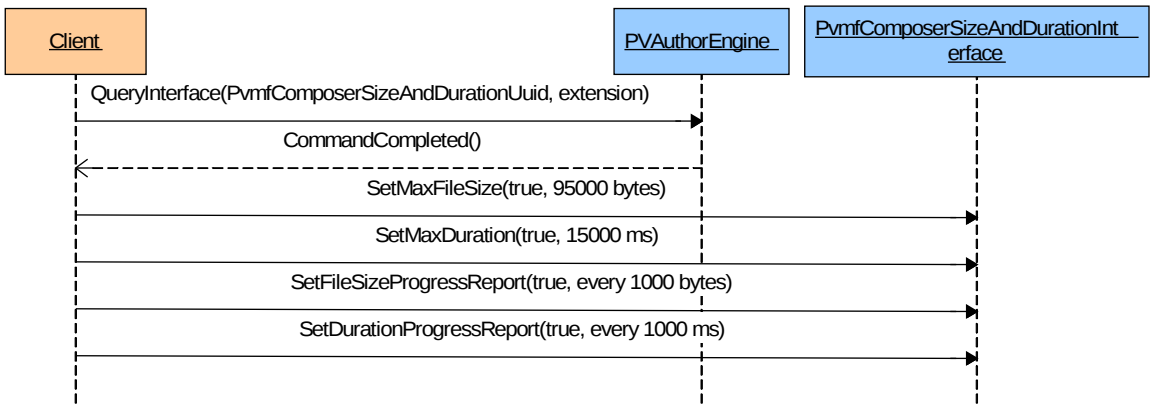

Figure 13: Max file size, duration and progress report configuration

After the authoring session is started, if the progress reports are enabled, PVAuthorEngine will send informational events to the Client at the specified frequency to the

PVInformationalEventObserver for the session. If the max file size or duration feature is enabled, when the file size or duration reaches the specified maximum, PVAuthorEngine will send an informational event to the Client after stopping the session and completed writing the output file. Figure 14 below illustrates the information events sent to the Client if the max file size, duration and progress report features are enabled.

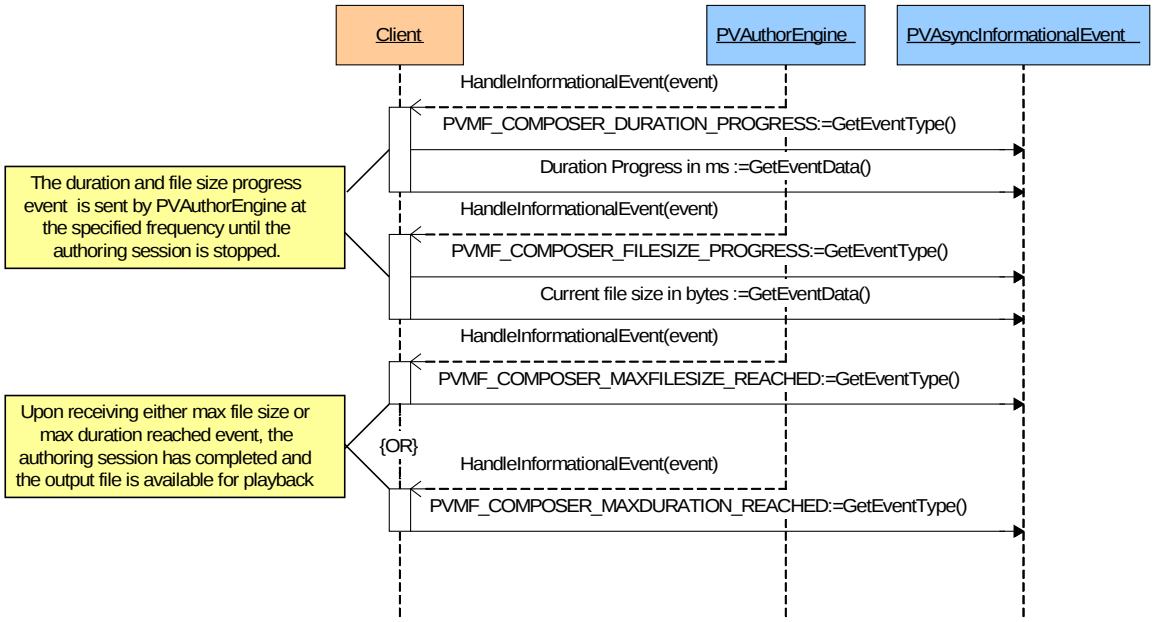

Figure 14: Informational events for progress report and max file size and duration

10. Initialize and Start Session

After setting up the session by selecting a file format composer, adding all media tracks and configuring all composers and encoders, the client can initialize and start the authoring session. Once Init is called, the settings and configuration of the authoring session will be set for the remainder of the session except special settings that are designated to be modifiable after PVAuthor engine is initialized or started. Otherwise, to modify the settings and configuration, the client will need to reset the authoring session and restart the session configuration process. PVAuthor engine will allocate resources for the session and connect to the data source capturing devices when it is being initialized. After PVAuthor engine is started, input data from various capturing devices will be encoded to the requested data formats, formatted to the requested file format and written out to file. Figure 15 below illustrates the sequence of calls to initialize and start the authoring session.

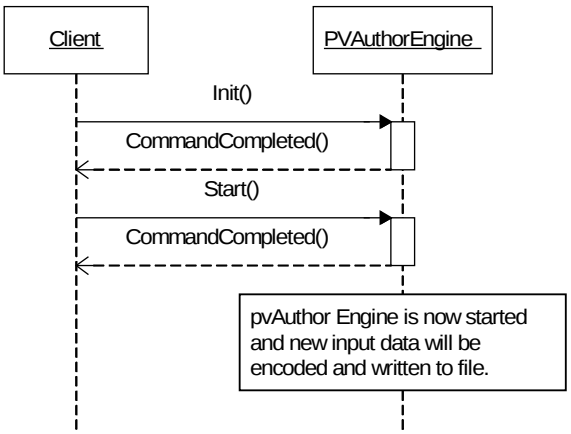

Figure 15: Initialize and start authoring session

11. Pause and Resume Session

After the authoring session is started, the client can pause the session if the application user chooses to pause the session or events such as an incoming call occurs. When PVAuthor engine is in paused state, it will not process any new input data from the capturing devices. However, if there are buffered input data captured before PVAuthor engine is paused, PVAuthor engine will continue to process the data and write to file until the buffered input is exhausted. The client can resume the authoring session any time after the session is successfully paused. After resume is complete, PVAuthor engine will resume processing input data from the capturing devices. Figure 16 below illustrates the sequence of calls to pause and resume an authoring session.

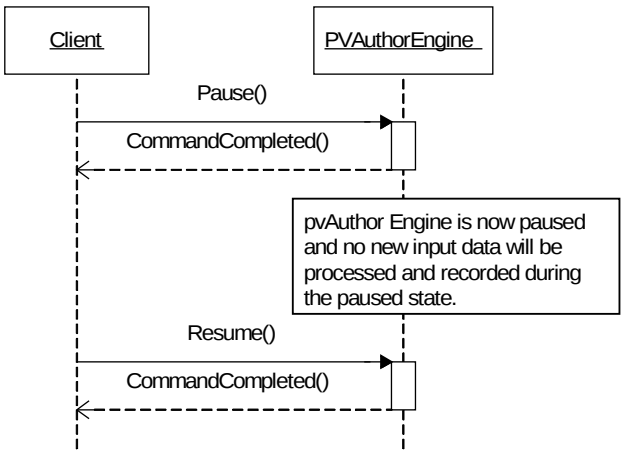

Figure 16: Pause and resume an authoring session

12. Stop Session

The client can stop the authoring session when it is in a started or paused state using the Stop method. If there is unencoded source data captured and buffered in the data path of PVAuthor engine, the default behavior is to encode them and add them to the output file, such that all media captured before the Stop will be written to the output file. This behavior can potentially cause a delay before the client would receive completion on the Stop call, depending on the amount of buffered up data and the processing power of the device. Figure 17 below illustrates the sequence of calls to stop an authoring session.

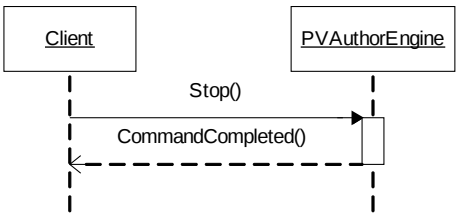

Figure 17: Stop authoring session

13. Reset and Close Session

When PVAuthor engine is in initialized, started or paused state, the Reset command would return PVAuthor engine to the opened state. If SelectComposer or AddMediaTrack were called, and PVAuthor engine returned configuration objects for the composer or media track, the client will need to call removeRef on the configuration object before Reset can be called. If PVAuthor engine is in started or paused state, it would first stop the authoring session. The session would then reset and the selected composer and media tracks for the session would be deleted. However, the added data source and sink nodes will remain available for PVAuthor engine to use. After Reset is complete, PVAuthor engine is returned to the open state and the client can call SelectComposer and AddMediaTrack to set up the session once again.

To close the session, the client should call RemoveDataSource to remove all data source previously added to PVAuthor engine and then call Close to close the session.

Figure 18 below illustrates the sequence of calls to reset and close an authoring session.

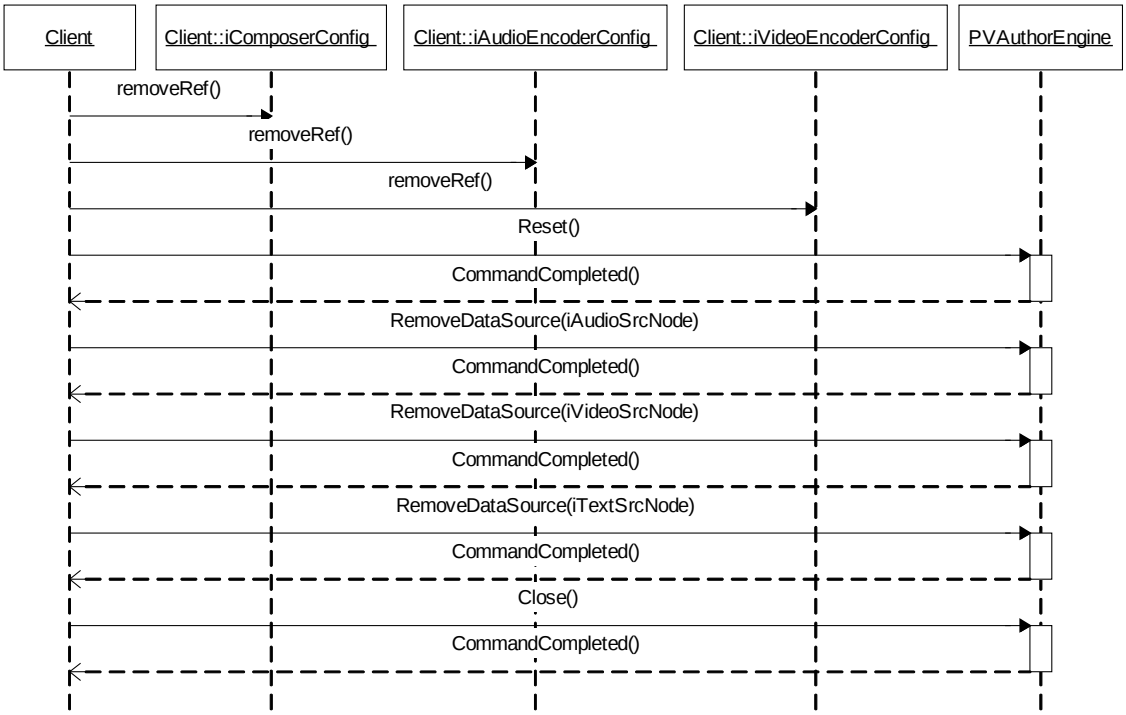

Figure 18: Reset and close authoring session

14. Capability Query and Configuring Settings

PVAuthor engine utilizes the PVMF capability-and-configuration interface to allow the application to access and modify engine and node settings not exposed by the author 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.

14.1. PVAuthor Engine Key Strings

All key strings at the PVAuthor engine level start with “x-pvmf/author”.

14.2. Node Level Key Strings

The node level key strings available during PVAuthor engine usage depends on PVMF nodes being used by the PVAuthor engine at that time and the key strings supported by a particular node. For node level key strings, PVAuthor engine acts as a router to pass any requests to the appropriate node. Currently, PVAuthor 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 PVAuthor engine.

Key Sub-String	Node Type
x-pvmf/video/render	Video Encode Node
x-pvmf/audio/render	Audio Encode Node
fileio/	Composer Node
x-pvmf/file/output	File Output Node
x-pvmf/media-io	Media Input Node
x-pvmf/avc/encoder	AVC Encoder

PVMF video and audio encoder node key strings are listed below. The key strings allow settings associated with M4v, H.263 and H.264 video encoding to be queried and modified when PVMF Video Encoder node is used to encode yuv bitstreams to mp4 or 3gp. The key strings are used by amr encoder to encode pcm data to mp4/3gp files.

Key Strings With Value Type	Description
x-pvmf/video/render/output_width;valtype=uint32	Set the output frame width
x-pvmf/video/render/output_height;valtype=uint32	Set the output frame height
x-pvmf/audio/render/sampling_rate;valtype=uint32	Set the sampling rate of audio bitstream
x-pvmf/audio/render/channels;valtype=uint32	Set the number of channels in audio bitstream
x-pvmf/avc/encoder/encoding_mode;valtype=uint32	Set the encoding mode

MP4 Composer Node key strings are listed below:

Key Strings With Value Type	Description
fileio/pv-cache-size	Set the file io cache size. This setting will depend on if the PV file cache

	setting has been turned-on in osclconfig_io.h
fileio/presentation-timescale	Set the timescale for overall 3GP presentation . Default value is 1000.

15. Error Handling in the PVAuthor Engine

The PVAuthor engine is responsible for mapping commands and requests from the application or higher-level components to potentially multiple steps and commands to the set of nodes/components under the PVAuthor engine's control. Errors may occur while processing a specific application request or may happen asynchronously outside of any request (e.g., an error may happen during the recording process after it has already started). The nodes/components attached to the PVAuthor engine report errors through the following methods:

1. HandleNodeErrorEvent for errors that happen asynchronously,
2. NodeUtilCommandCompleted for errors that happen while processing a specific request.

If any errors happen while the PVAuthor engine is processing a request from the application, PVAuthor will wait for any pending commands it has to underlying nodes/components and then return the error information as part of the CommandComplete status. If the errors happen outside of any request or command from the application, the information will be sent using the HandleErrorEvent method. The idea of collecting even the asynchronous errors that happen during the processing of a request and sending it back during CommandComplete is to avoid reporting the same errors multiple times. The PVAuthor engine will transition to an error state when it reports these errors and the application can call Reset as described in Section 13 to recover. Figures 19 and 20 show the API sequences and the logic flowchart for the error handling described here.

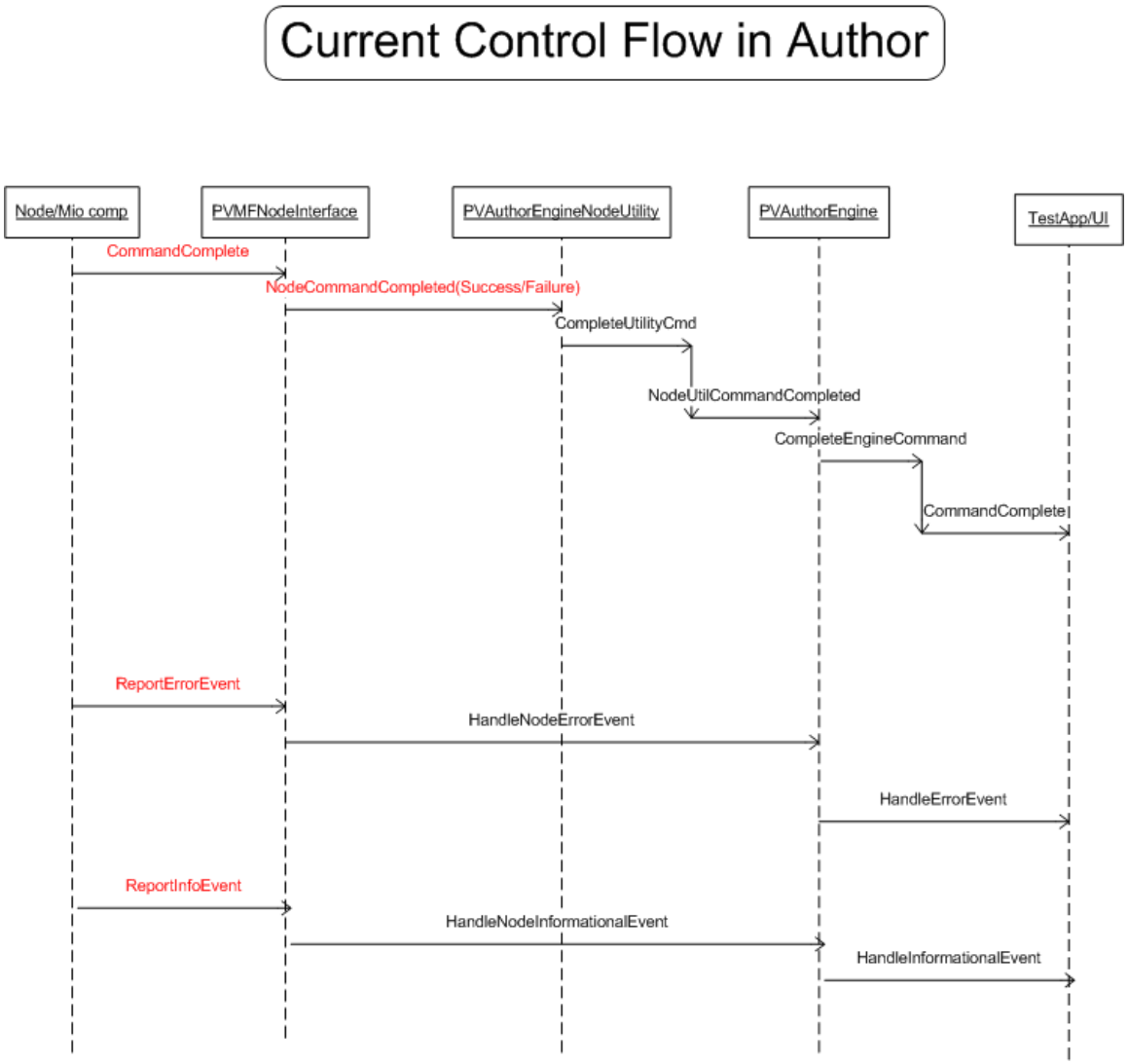

Figure 19: Propagation of error information.

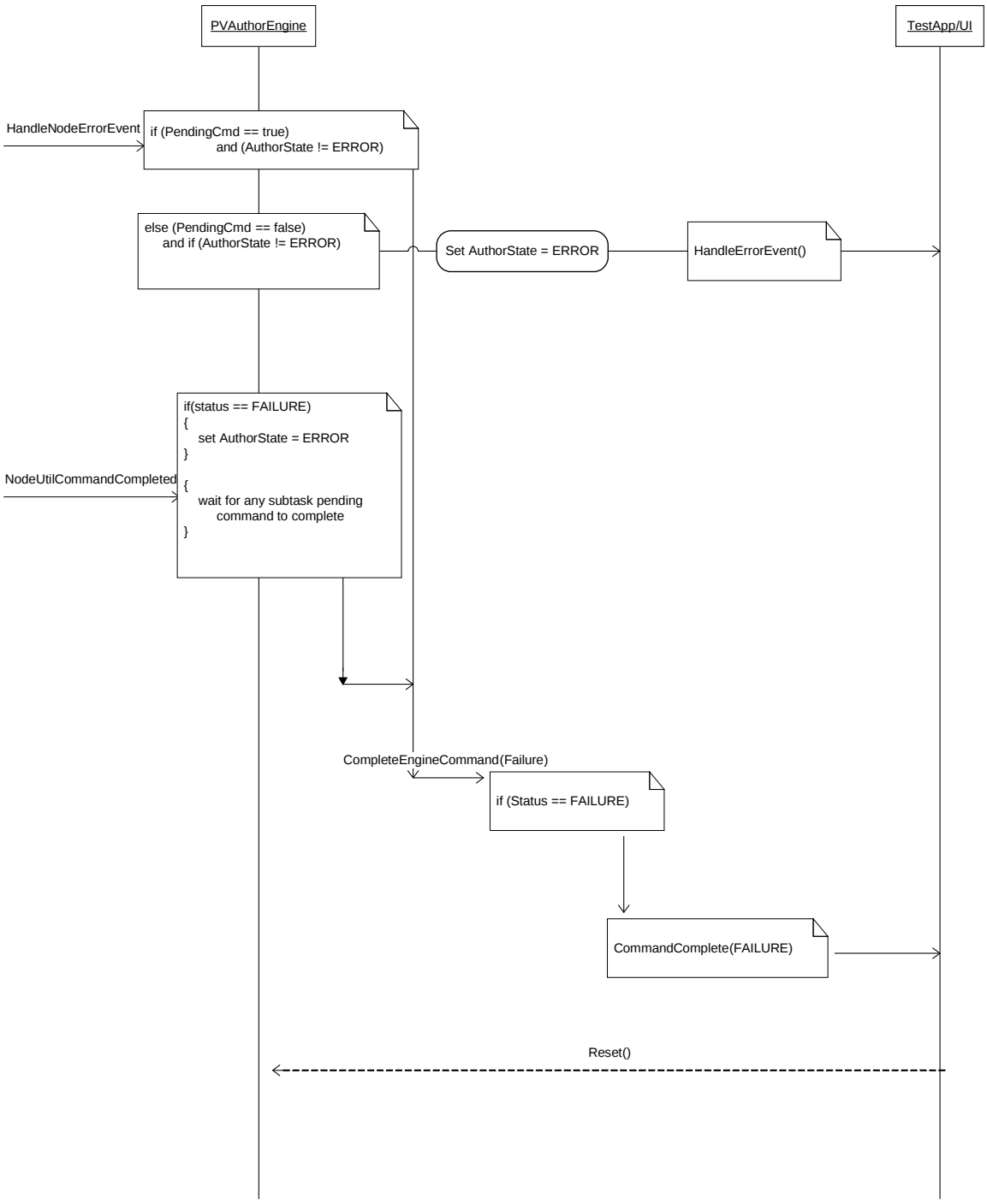

Figure 20: PVAuthor error handling flowchart.