Primetime Recording Server 1.0

Getting Started Guide

Contents

Primetime Recording Server workflow 3
Ingesting live stream (HDS) 4
Ingesting live stream (HLS) 4
Removing slates 5
Multi-tenancy 5
Recording Container 5
Recording Module 5
REST APIs 6
Recording Module REST API 7
Task REST API 8
Authentication 10
Set level manifest/playlist 10
Uploading recorded clips 11
HTTP Uploading 11
DRM/FAXS 11
Configuration for a Recording Module 12
Interface for access plugins 14
com.adobe.fms.util.httpfileaccess.SimpleHttpAccess plugin 14
com.adobe.fms.util.httpfileaccess.PTOriginAccess plugin 14
Interface to create custom access plugins 15
Configuration for a Task Module 16
Logging 21
Logging Namespace 21
Full Event Replay 22

Primetime Recording Server

The Primetime Recording Server component in a Live2VoD solution can record live streaming events, remove slates, and create and publish clipped highlights or full-event replays for later viewing.

Primetime Recording Server can also record streams from non-Primetime sources, by acting as a client for HDS and HLS live streams. This way, it can leverage the redundancy and fault resilience in the live publishing system.

The primary function of Primetime Recording Server is to create and upload the clips. To implement the complete Live2VoD workflow, use proprietary or third-party solutions to publish clip metadata to a CMS or publish the VOD AdMap to the AD server.

Primetime Recording Server workflow

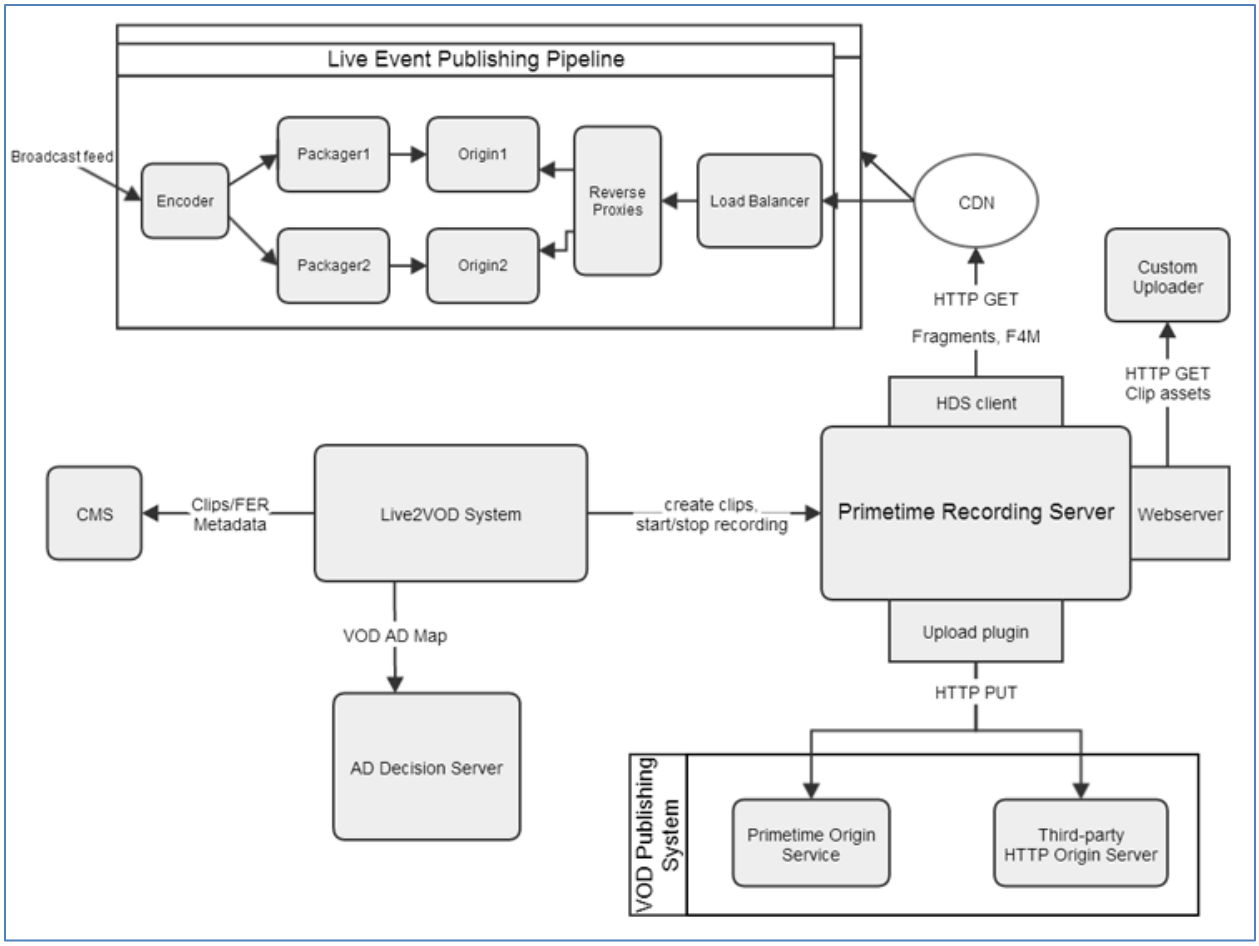
A typical Primetime Recording Server workflow comprises the following tasks required to archive live video assets and upload them for playback:

1. Ingest live stream: Primetime Recording Server can ingest live stream from any HDS or HLS source to create clips and Full-Event-Replay. For example, Primetime Recording Server can ingest live feed from Primetime Origin, AMS, Primetime enabled encoder, or any non-Primetime packaging source.

2. (Optional) Remove slates: Slates may be specified explicitly or indicated using in-manifest cues. Primetime Recording Server can optionally re-base clipped media timestamps to remove gaps that appear after removing slates. It can retain in-manifest cues or align the ends of slates with a keyframe.

3. Upload archived assets: Primetime Recording Server can upload archived assets to any target location using either HTTP or the File System write command. If Primetime Recording Server uses the File System write command to upload assets, the target should either be a local disk or be accessible through a local mount point, for example NFS mount point.
   Primetime Recording Server can upload assets to the live setup where clip-manifests reference the same set of the fragments as those for the live publishing system. Alternatively, it can upload assets to a different VOD setup, which may comprise Primetime servers. Primetime Recording Server uses HTTP to either upload assets as soon as it fetches a fragment or at the end of the recording.

The following diagram depicts the workflow that Primetime Recording Server follows to record a live video stream for on-demand playback:



Ingesting live stream (HDS)

To create a clip, Primetime Recording Server creates an instance of the HDS client. By default, the Best-effort-fetch functionality is enabled for each instance of the client. The bootstrap of a clip contains a fragment when either it is referenced in the source live bootstrap or the fragment can be obtained using the BEF logic. Gaps caused by missing fragments are removed from the bootstrap by rebasing the content that follows. To enable a clip to start at a keyframe, set the value of the `BreakAtKeyFrame` parameter to `True`. If the specified start time doesn't coincide with a keyframe, then it is adjusted to the next or the previous keyframe.

Ingesting live stream (HLS)

To create a clip, Primetime Recording Server creates an instance of HLS client. The HLS client fetches segments referenced in the source live playlist. It caters to a stream-level level manifest only. For HLS recording, in the current version, you cannot specify the start and end times for a clip. Clips can be created starting from the first obtainable segment or the third-last segment in the source playlist. Clips end at the end of a playlist. Alternative, you can end a clip using the stop task API.

Removing slates

The HDS/HLS client supports the removal of slates that are specified as pairs of start and end timestamps. The HDS client parses ad cues and removes the content within the cue duration, if it is indicated that the Ad cues in the live stream mark the slates. If slate removal is requested, the fragments/segments within the slate are not recorded. For HDS, the fragments overlapping with slate boundary are recorded with data that lies outside the slate. Appropriate gaps (discontinuities) are inserted in the recorded bootstrap/M3U8 corresponding to the slates when re-basing is disabled. When rebasing is enabled, the content of the fragment that contains the end of a slate is merged with the fragment that contains the start of a slate. In addition, the content is rebased to remove the gap in the timestamps between start and end times of the slate.

Multi-tenancy

The HDS configuration is structured at both the container and recording module levels. You can create several recording tasks within a single recording module.

Recording Container

The container module provides multi-tenancy using authentication parameters for the REST API. You can configure these parameters at the container module. The container-level configuration contains shared secrets for the authentication of requests to manage clips, for example creating recording requests, uploading clips, and fetching clip data. You can access the REST API on the recording modules and recording tasks under the container using authentication parameters specified inside the container. This way, you can configure a separate container for each tenant.

Recording Module

You can leave the configuration for the recording module empty. In this case, the task creation REST API should include the configuration parameter.

Alternatively, you can specify common configuration profiles within the `recording.xml` file. This file is configured on the server in the conf folder within the Recording Module. Ensure that he file is created or updated at the server only and is not accessible using REST APIs. The stored configuration parameters in the `recording.xml` file are called Recording Profiles. These are canned configuration that can be referenced in task creation REST API (for moe information, see Task REST API).

The following are the advantages of having canned configuration on the server:

- Profiles help avoid repetition of commonly used parameters.
- Profiles enable you to keep secret options (for example, shared secret for authenticating uploads) in the configuration file, which is not visible/updated through REST APIs. For example, credentials for Primetime Origin where recorded assets are to be pushed can be kept in `recording.xml` file. The task creation API only refers to the stored configuration and credentials need not be exchanged over a network.

REST APIs

Primetime Recording Server hosts a REST interface to facilitate live recording and clipping workflows. The interface contains APIs to create new recording tasks for a given recording module. The APIs also provide modify options (end-time), options to fetch or delete the status of an existing recording task. You can use the APIs to enumerate all recording tasks for a given recording-request URL, upload recorded files, or fetch a list of files for a recording.

A recording task has the following attributes:

- Start and end time. These timestamps can be one of the following:
    - Stream timestamps as indicated in the bootstrap (applies to HDS ingest only)
    - Time in SMPTE timecode format (applies to HDS ingest only)
    - Offsets. For HDS, you can specify the start time as an offset from the beginning of the stream. Similarly, you can specify the end time as an offset from the start timestamp (duration for the recording). To record from the first obtainable fragment, specify the start time offset as zero. For HLS ingest, specify 0 as the value start time offset to record from the first obtainable segment. The End time offset is not valid for HLS ingest. HLS recording stops when it encounters the end of presentation or when the recording task is stopped using a REST API.
    - If all the attributes are omitted, recording starts from the current media time in the live bootstrap and continues as far as possible for HDS. Recording ends when an END_OF_PRESENTATION discontinuity is encountered if the end-time is not provided or continues until 24 hours (whichever is earlier). For HLS, recording starts from the third-last segment available in the source playlist/m3u8/ and continues as far as possible. The recording stops when it encounters the end of presentation, if end-time is not specified. Otherwise, the recoding stops when the recording task is stopped using a REST API.
- Source manifest/playlist URL
- Flag indicating whether the fragments should also be recorded/uploaded. The default value is true. If the value is false, the recording task only uploads a clip manifest/playlist and not the fragments. This feature is useful when multiple clips from a single stream overlap and reference the same set of the fragments.
- For HDS, fragments, manifests, and DRM are recorded using a specified prefix. For HLS recording, the prefix is not applied to the recorded segments.
- The relative path of the fragments/segments with respect to the manifest/playlist.
- Adjust the clip-start and the slate-end-times to match the keyframes (HDS only). A flag indicates whether additional content is to be added or removed from the clip. This functionality is disabled by default.
- Retain in-manifest cues (disabled by default)
- Optional slate removal, enabled by default. In Manifest cues, this attribute indicate slates. If slate removal is enabled, in-manifest cues are not retained.

Recording Module REST API

For the recording module r_module configured on the container r_cont, the REST API request URL is the following:

```
http://<recording server address: port>/r_cont/r_module
```

Http Request Method	Details

GET	Returns an xml file that contains a list of tasks configured on the module. The list only contains the top level tasks. If any task is a set-level recording task, the URI for each individual rendition recording is not included in the xml file. URIs for the rendition level recordings are returned when you perform a GET operation on the set level task URI.
	Sample Request
	`http://<recording server address: port>/r_cont/r_module`
	Sample Response
	`<tasks>`
	`<task uri="/r_cont/r_module/taskID1" />`
	`<task uri="/r_cont/r_module/taskID2" />`
	`</tasks>`

POST	Creates a new task. For details on various fields of the xml request, see the configuration of the task.xml file.
	Sample Request
	`<Config>`
	`<URL>http://localhost/hds/artbeats.f4m</URL>`
	`<RecordingPrefix>artbeats</RecordingPrefix>`
	`<StartTime>0</StartTime>`
	`<StartTimeOffset>0</StartTimeOffset>`
	`<EndTimeOffset>20000</EndTimeOffset>`
	`<ManifestTargetID>target_origin</ManifestTargetID>`
	`<ProfileID>profile1</ProfileID>`
	`<Targets>`
	`<Target id='1'>`
	`<Type>disk</Type>`

Http Request Method	Details
	<Path>D:/test</Path> </Target> </Targets> </Config> **Sample Response** If the task is created, HTTP response code 201(CREATED) is returned. The URI for task REST API is sent in the Location header.

Task REST API

For a task with id task_UUID created inside a recording module r_module within the container r_cont, the REST API request URL is the following:

`http://<recording server address: port>/r_cont/r_module/task_UUID`

Http Request Method	Details

GET	Returns an XML file containing URIs that allow further state management for the task. If this is a set-level recording task, the URIs for the child tasks is also returned.
	
	Sample Request
	`http://<recording server address: port>/r_cont/r_module/task_UUID`
	
	Sample Response
	`<task>`
	`<state desc="Status"`
	`uri="/test/event_a/927ff3bfa201417e8ee790161c96e7c5/state"/>`
	`<state desc="Statistics"`
	`uri="/test/event_a/927ff3bfa201417e8ee790161c96e7c5/stats"/>`
	`<state desc="Config"`
	`uri="/test/event_a/927ff3bfa201417e8ee790161c96e7c5/config"/>`
	`<tasks>`
	`<task`
	`uri="/test/event_a/927ff3bfa201417e8ee790161c96e7c5/d20b149c35474d838b`
	`d56c3721b9693d"/>`
	`<task`
	`uri="/test/event_a/927ff3bfa201417e8ee790161c96e7c5/ebff3641cf9a427a90`
	`db5e1cb69a01e5"/>`
	`</tasks>`
	`</task>`

Http Request Method	Details

GET on /stats	Returns an XML containing task statistics.
	
	Sample Request
	`http://<recording server address: port>/r_cont/r_module/task_UUID/stats`
	
	Sample Response
	`<stats>`
	`<stat name="Total fragments downloaded">`
	`<desc>Property Total fragments downloaded</desc>`
	`<statType>long</statType>`
	`<value>14</value>`
	`</stat>`
	`<stat name="Last fragment downloaded">`
	`<desc>Property Last fragment downloaded</desc>`
	`<statType>long</statType>`
	`<value>69</value>`
	`</stat>`
	`<stat name="Total fragments uploaded">`
	`<desc>Property Total fragments uploaded</desc>`
	`<statType>long</statType>`
	`<value>14</value>`
	`</stat>`
	`<stat name="Last fragment uploaded">`
	`<desc>Property Last fragment uploaded</desc>`
	`<statType>long</statType>`
	`<value>69</value>`
	`</stat>`
	`<stat name="Current recording time">`
	`<desc>Property Current recording time</desc>`
	`<statType>long</statType>`
	`<value>276024</value>`
	`</stat>`
	`<stat name="Download error count">`
	`<desc>Property Download error count</desc>`
	`<statType>long</statType>`
	`<value>0</value>`
	`</stat>`
	`<stat name="Recording error count">`
	`<desc>Property Recording error count</desc>`
	`<statType>long</statType>`
	`<value>0</value>`
	`</stat>`
	`</stats>`

POST on /stats	Not supported. Returns BAD REQUEST (400) **Sample Request** `http://<recording server address: port>/r_cont/r_module/task_UUID/stats` **Sample Response**

Http Request Method	Details
	BAD REQUEST (400)
GET on /state	Not supported currently. Will be supported in next version **Sample Request** `http://<recording server address: port>/r_cont/r_module/task_UUID/state` **Sample Response** BAD REQUEST (400)

POST on /state/stop	State can be updated. The new state is specified as path component of URI. Currently, only changing the state to stop is supported. This feature stops the task. However, the task state is not deleted from disk. When the server restarts, the tasks do not start again. To completely remove the tasks, deleted them from the disk. For set level task, all the child tasks are stopped before the set level task. This stop operation runs asynchronously. This implies that the stop command is issued to the task worker and then a response is sent to the user. Task worker completes its current processing tasks and then stops recording further.
	Sample Request
	`http://<recording server address: port>/r_cont/r_module/task_UUID/state/stop`
	Sample Response
	`CONFLICT (409) is returned if the task has already been stopped. OK (200) is returned if the stop command is successfully submitted to the task worker.`

Authentication

REST interface requests are authenticated using the SecurityToken mechanism. The shared secret is configured in container.xml file.

Set level manifest/playlist

The source manifest/playlist URL in the create task request can be a set-level manifest/playlist. This feature enables you to record multiple stream renditions simultaneously and generate a set-level manifest/playlist for the recording. The following points apply for a clip creation task that uses a set-level manifest/playlist:

- The adjustment of clip-start and slate-end with keyframe parameter is ignored and the feature is considered to be disabled.
- Recording prefix / fragment-relative-path is not included in the request and ignored otherwise.

- An instance of recording task is created for each contained rendition stream. The status is queried and deleted separately. Recording tasks for individual renditions are created as child tasks of the set-level recording task. Deleting or stopping the set level recording task deletes the child tasks.
- The relative path and filenames of the rendition manifests and referenced fragments are generated automatically.
- The set-level manifest/playlist is fetched only once when a task is created for each of the referenced stream level manifests/playlists. Any subsequent update to the set-level manifest/playlist is not handled.
- No synchronization is performed across stream-level recording tasks created from a set level manifest/playlist. Stream level tasks spawned from the set level manifest/playlist progress independently without sharing their state.
- The multi-level manifest format (specified in f4m 2.0 specifications) is supported. Therefore, media elements in the set-level manifest refer to the stream level f4m.
- Adaptive sets in set-level manifest are not supported. In addition, groups are not supported for HLS.

Uploading recorded clips

The recorded assets are uploaded to the location specified in the recording clip creation request. By default, you can specify targets only in the recording module on the server. In addition, a recording request can only point to a pre-configured target in the recording module.

To enable upload targets to be specified in the recording request, set the optional configuration option `AllowCustomTargets` in the `recording.xml` file of the recording module to true. By default, this option is set to false.

HTTP Uploading

Primetime Recording Server provides a plugin interface to optionally secure HTTP PUT upload requests. The plugin can add custom headers with tokens in the HTTP PUT request for authentication or modify the PUT request URL. For details of the interface to upload plugins, see Interface to create custom access plugins.
Recording Server comes with two default plugin implementations

- To upload files to Primetime origin server.
- To upload files to a standard http server without any authentication

To know about the configuration and other details about the default plugins, see Interface for access plugins.

DRM/FAXS

If the source HDS/HLS live stream is encrypted, the recorded fragments are also encrypted. The recorded DRM metadata and DRM policies remain unchanged. If the source f4m/m3u8 contains DRM Metadata inline, the recorded f4m/ m3u8 also has inline DRM Metadata. Otherwise, the DRM Metadata is written to the same target as the recorded f4m/m3u8.

Configuration for a Recording Module

Config Name	Type	Description
//Config/AllowCustomTargets	String	Can assume values either true or false. This configuration lets you specify the upload target at the time of task creation. By default, this config is assumed false. When it is set to false, you can only upload recorded assets to one of the pre-configured targets specified in the recording.xml profiles.
//Config/Profiles/Profile	XML element	The root element for a recording profile
//Config/Profiles/Profile/@id	String	A unique alphanumeric ID to identify the Recording Profile. This ID will be used by the task configuration to reference the profile.
//Config/Profiles/Profile/Targets	XML element	List of possible targets to save the recorded assets(fragments, metadata, manifest).
//Config/Profiles/Profile/Target/Target/@id	String	A unique alphanumeric ID to identify the target.
//Config/Profiles/Profile/Target/Target/Type	String	Type of target. Currently only two types are supported - "http" and "disk"
//Config/Profiles/Profile/Target/Target/AccessPlugin	XML Element	This is used to specify the access plugin which can be used to authenticate http requests to the specified http target. The configuration and access plugin

Config Name	Type	Description
		interface are same as those used to authenticate the http requests for mp4 in Smart Origin
//Config/Profiles/Profile/Target/Target/Path	String	For target of type "http", the disk path where the recording assets must be written.

The following is a sample recoding.xml file for the above configuration:

```
<Config>
<!--
Uncomment this configuration to allow user to specify custom upload target.
By default this config is assumed false. When set false, user can only upload recorded assets
to one of the pre-configured targets specified below.

    <AllowCustomTargets>true</AllowCustomTargets>
-->
    <Profiles>
        <Profile id='profile1'>
            <Targets>
                 <Target id='target_basehttp'>
                    <Type>http</Type>
                    <AccessPlugin>
                        <Class>com.adobe.fms.util.httpfileaccess.SimpleHttpAccess</Class>
                        <Init>
                            <BaseURL>http://host:port/root_path</BaseURL>
                        </Init>
                    </AccessPlugin>
                </Target>
                <Target id='target_origin'>
                    <Type>http</Type>
                    <AccessPlugin>
                        <Class>com.adobe.fms.util.httpfileaccess.PTOriginAccessPlugin</Class>
                        <Init>
                            <URL>http://origin_host/_default_/_default_</URL>
                            <UseSecurityToken>true</UseSecurityToken>

<SecurityTokenKey>4ff4756ed68239d34d482dbc88819abc</SecurityTokenKey>
                        </Init>
                    </AccessPlugin>
                </Target>
                <Target id='target_local_disk'>
                    <Type>disk</Type>
                    <Path>z:/file_path</Path>
                </Target>
            </Targets>
        </Profile>
    </Profiles>
</Config>
```

Interface for access plugins

Primetime Recording Server provides the following default access plugins:

- com.adobe.fms.util.httpfileaccess.SimpleHttpAccess
- com.adobe.fms.util.httpfileaccess.PTOriginAccessPlugin

com.adobe.fms.util.httpfileaccess.SimpleHttpAccess plugin

This plugin provides simple access to an HTTP server without requiring any authentication.

The following is the configuration specification for the plugin:

Configuration	Description
//Init/BaseURL	Base URL of the Http Server

Here is a sample configuration for the plugin:

```
<Targets>
                <Target id='target_basehttp'>
                   <Type>http</Type>
                   <Timeout>5000</Timeout> <!-- Connection timeout in milliseconds -->
                   <AccessPlugin>

   <Class>com.adobe.fms.util.httpfileaccess.SimpleHttpAccess</Class>
                                  <Init>
                                         <BaseURL>http://host:port/root_path</BaseURL>
                                  </Init>
                            </AccessPlugin>
               </Target>
</Targets>
```

com.adobe.fms.util.httpfileaccess.PTOriginAccess plugin

This plugin supports upload of VOD assets to Primetime Origin

The following is the configuration specification for the plugin:

Configuration	Description
//Init/URL	VOD Module URL on Primetime Origin to which the assets must be published
//Init/UseSecurityToken	True if auth token is enabled on Primetime Origin, False otherwise
//Init/SecurityTokenKey	SecurityToken for VOD module on PT Origin

Here is a sample configuration for the plugin:

```
xml
 <Target id='target_origin'>
           <Type>http</Type>
          <AccessPlugin>
```

```
   <Class>com.adobe.fms.util.httpfileaccess.PTOriginAccessPlugin</Class>
                  <Init>
                         <URL>http://origin_host/_default_/_vod_</URL>
                         <UseSecurityToken>true</UseSecurityToken>

   <SecurityTokenKey>4ff4756ed68239d34d482dbc88819abc</SecurityTokenKey>
                  </Init>
            </AccessPlugin>
</Target>
```

Interface to create custom access plugins

Primetime Recording Server provides the `com.adobe.fms.util.httpfileaccess.HttpAccessPlugin` interface to let users create custom plugins to upload the recorded VOD assets on an HTTP server.

The following is the sample implementation of the interface:

```
**************************************************************************/
package com.adobe.fms.util.httpfileaccess;
import java.util.logging.Logger;
import org.w3c.dom.Element;
/**
 * Plugin instances should be thread safe as same instance will be
 * shared by multiple http file readers.
 **/
public interface HttpAccessPlugin {
   /**
    * Hook to initialize the plug-in, passing the &lt;init&gt; element within
    * the HttpAccessPlugin config element or <tt>null</tt> if no initialization
    * configuration was provided.
    * This method is invoked during module initialization and is guaranteed
    * to have finished execution before any calls to {@link #authenticateRequest(HttpGet)}
    * are made.
    *
    * @param configXML The &lt;init&gt; child element of the plug-in config
    * element.
    * @param logger Logger to be used for logging
    */
   public void initialize(Element configXML, Logger logger)
   throws Exception;

   public void authenticateRequest(HttpRequest httpRequest, String resourceURI)
   throws Exception;
}
```

The following is the Java code for the com.adobe.fms.util.httpfileaccess.HttpRequest Interface:

```
package com.adobe.fms.util.httpfileaccess;
import java.util.List;
import java.util.Map;
public interface HttpRequest {
    /**
    * Returns the URI this request uses, such as http://example.org/path/to/file.
    * Note that the URI may be absolute URI (as above) or may be a relative URI.
    */
   public String getURI();

   public void setURI(String uri);
```

```
   /**
    * Returns the HTTP method this request uses, such as GET,
    * PUT, POST, or other.
    */
   String getMethod();

   /**
     * Returns the header value with the specified header name.  If there are
     * more than one header value for the specified header name, the first
     * value is returned.
     *
     * @return the header value or {@code null} if there is no such header
     *
     */
    String getHeader(String name);
    /**
     * Returns the header values with the specified header name.
     *
     * @return the {@link List} of header values.  An empty list if there is no
     *         such header.
     */
    List<String> getHeaders(String name);
    /**
     * Returns the all header names and values that this message contains.
     *
     * @return the {@link List} of the header name-value pairs.  An empty list
     *         if there is no header in this message.
     */
    List<Map.Entry<String, String>> getHeaders();
    /**
     * Returns {@code true} if and only if there is a header with the specified
     * header name.
     */
    boolean containsHeader(String name);
    /**
     * Sets the content of this message.
     */
    void setContent(byte[] content);
    /**
     * Adds a new header with the specified name and value.
     */
    void addHeader(String name, String value);
    /**
     * Sets a new header with the specified name and value.  If there is an
     * existing header with the same name, the existing header is removed.
     */
    void setHeader(String name, String value);
    /**
     * Removes all the header with the specified name.
     */
    void removeHeader(String name);
    /**
     * Removes all headers from this message.
     */
    void clearHeaders();
}
```

Configuration for a Task Module

Name	Description	DataType	Type
TimeInSmpte	(HDS only) Indicates if "StartTime" and "EndTime", if given, are in SMPTE TimeCode format.	Boolean	Optional. Default value: false
StartTime	(HDS only) If TimeInSmpte is set to true, then this should be repackaging start time in the format: HH:MM:SS:FF,<date>,<timezone>, where date and timezone are optional and FF corresponds to number of frames. **Note**: Currently, fps is assumed to be 30 and NTSC drop frames/other fps are not considered. If TimeInSmpte is set to false, then this should be the actual live stream time-stamp as indicated in the bootstrap.	String, if in SMPTE format, else Integer	This field can be omitted, and startTimeOffset, if given, will be considered. If both StartTime and StartTimeOffset are omitted, the recording starts from the current media time in the live stream bootstrap (equivalent of start-recording-now)
EndTime	(HDS only) If TimeInSmpte is set to true, then this should be repackaging start time in the format: HH:MM:SS:FF,<date>,<timezone>, where date and timezone are optional and FF corresponds to number of frames. Note: Right now fps is	String, if in SMPTE format, else Integer	This field can be omitted, and EndTimeOffset, if given, will be considered. If both EndTime andEndTimeOffset are omitted, the recording continues as far as possible. It will end on encountering END_OF_PRESENTATION discontinuity when end-time is not provided or will continue for 24 hours (whichever is

	assumed to be 30 and NTSC drop frames/other fps are not considered. If TimeInSmpte is set to false, then this should be the actual live stream time-stamp as indicated in the bootstrap.		earlier)
StartTimeOffset	(HDS only) Repackaging start time given as an offset from the beginning of the stream. To start recording from the first obtainable fragment, start time offset should be given as zero.	Integer	This field can be omitted, and StartTime, if given, will be considered. If both StartTime and StartTimeOffset are omitted, then the recording will start from the current media time in the live stream' bootstrap (equivalent of start-recording-now)
EndTimeOffset	(HDS only) End time offset given as an offset from the repackaging start time-stamp (i.e. duration for the recording).	Integer	This field can be omitted, and EndTimeOffset, if given, will be considered. If both EndTime and EndTimeOffset are omitted, the recording will extend as far as possible. It will end on encountering END_OF_PRESENTATION discontinuity when end-time is not provided or will continue for 24 hours (whichever is earlier)
RecordFragments	Option to specify if fragments/segments are to be recorded. When it is false, then the recording task will only create and upload a clip manifest/playlist and not the fragments. It will be useful for the scenario where multiple clips, created from a single	Boolean	Optional. Default value: true

	stream, are overlapping and they reference the same set of the fragments.		
FragRelativePath	Relative path for the fragments. If RecordFragments is set to true, this is the path, relative to the manifest/playlist path, where fragments will be uploaded.	String	Optional. Default value: ""
RecordingPrefix	Prefix to be used for recorded contents.	String	Mandatory
BreakAtKeyFrame	(HDS only) Option to specify if clip-start and the slate-end-times should be adjusted to match with keyframes.	Boolean	Optional. Default value: false
PackageMoreData	(HDS only) If BreakAtKeyFrame, this will indicate whether more content will be added to the clip or removed from it for the key-frame adjustment. If set to true, and if clip-start/slate end does not coincide with a key-frame, then the times will be adjusted to the immediate previous keyframe. If set to false, times will be adjusted to the immediate next keyframe.	Boolean	Optional. Default value: false
RetainCues	Option to specify if in-	Boolean	Optional. Default value: false

	manifest cues should be retained in the recording manifest.		
SlateRemoval	Option to specify if slates are to be removed. In-manifest cues as in the source manifest will indicate slates. If this is set to true, in-manifest cues won't be retained in the recording manifest.	Boolean	Optional. Default value: true
EnableBestEffort	(HDS only) Option to turn on/off best effort fetch.	Boolean	Optional. Default value: true
BlindFetchMaxFwdFetches	(HDS only) This is the maximum number of consecutive failed forward fetches allowed before reverting to the non-best-effort behavior.	Integer	Optional. Default value: 2
BlindFetchMaxBackFetches	(HDS only) This is the maximum number of consecutive failed backward fetches allowed before reverting to the non-best-effort behavior.	Integer	Optional. Default value: 2
BlindFetchFragDuration	(HDS only) The ideal duration of a fragment in live stream, in millisecs.	Integer	Optional. Default value: 4000 ms
BlindFetchSegDuration	(HDS only) The ideal duration of a segment in live stream, in millisecs.	Integer	Optional. Default value: -1, indicating 1 segment
ProfileID	ID of the profile from recording.xml from where default configurations should be picked. Refer//Config/Profiles/Profile/@id config in the	String	It's an optional parameter. It needs to be specified only if the task needs to pick certain default values from profiles specified in recording.xml

	config section of the recording.xml file.		
Targets	List of possible targets exactly similar to the way specified in recording.xml	XML Node	Refer to Targets config in recording.xml **Note**: You can specify it only if the value of the `AllowCustomTargets` configuration parameter is set to true in the recording module.
ManifestTargetID	Target ID to which manifest should be pushed. The target ID can point to any of the targets in the task.xml or in recording.xml. Targets from recording.xml can only be used if ProfileID value is set to point to a recording profile	String	It's a mandatory parameter. Refer to//Config/Profiles/Profile/Target/Target/@idconfig in recording.xml

Logging

Primetime Recording Server uses `java.util.logging` for logging. You can configure logging using the properties file `logging.properties` in the install directory. The scripts `rserv_start.bat` and `rserv_start.sh` load the `logging.properties` file from the install directory. The default file output is stored in the logs directory under the install directory.

Logging Namespace

Name	Description
com.adobe.fms	The root namespace for all Recoding Server logging. Output is defaultly set to ./logs
com.adobe.fms.http	Logging for the built in http server. Logs all requests and responses. Output is defaulty set to ./logs/http/access.log

The logging output from the recording module level is prefixed with `containerName,recordingModuleName`.
The logging output from the container level is prefixed with `containerName`.

Full Event Replay

After a hosted event completes, you can archive the content as a full event replay (FER). To create FER recording, exclude the end time from the options you specify when you record an event.

In this case, the recording continues until the event completes. Recording automatically stops when the live stream ends. You can also update the recording task later to specify the end time of the recording. The slate removal feature of Primetime Streaming Server lets you ignore live Ad cues in the FER and use an Ad map provided by an Ad server to change Ad load dynamically. For more information, see Removing slates.

Primetime Recording Server

Edition 1.0

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.