Live Profiler API

Generated on April 08,2021

Status: release 1.0.7

Contents

1 Main Page

DISCLOSURE. *This documentation contains ContentArmor (*http://www.contentarmor.net*) proprietary and confidential information. Passing on and copying of this document, use, extraction and communication of its content, is not permitted without written authorization from ContentArmor.*

ContentArmor Live Profiler is one of the system components of ContentArmor Video Watermarking system. It is responsible for preprocessing live encoded video feeds in preparation for watermarking embedding at a later stage in the content delivery pipeline. It is delivered as a software library compiled for a target platform e.g. CentOS 7.0 or Debian Wheezy. The library can be integrated either with an interface at the elementary stream (ES) level or at the transport stream (TS) level using slightly different APIs. Strictly speaking, the TS library is a wrapper on top of the ES library. Moreover, the ES interface provides a finer control on latency.

1.1 Operation Modes

ContentArmor Live Profiler library can operate in different modes depending on the deployment scenario. These operation modes currently include:

BROADCAST: the library introduces proprietary metadata as Supplemental Enhancement Information (SEI) Network Abstraction Layer Units (NALU) incorporated in the ES access units (AUs) of the video feed. The watermarking pace is set according to the number of marks found in the content. ContentArmor Embedder shall be placed further downstream to embeded the desired payload in the video. This mode is currently incompatible with Adaptive Bit Rate (ABR) content.

AB: the library generates two prewatermarked versions of the video stream (version A and version B) for each input video stream of an ABR bundle. Another component has to be placed further downstream to create a unique sequence of A-B video segments in order to encode a payload, e.g. using playlist manipulation or redirection at the edge.

ABJIT: the library incorporates SEI NALUs to the ES AUs and the watermarking pace is set according to the ABR video fragmentation. ContentArmor Embedder shall be placed further downstream (typically in a Packager) to generate the prewatermarked A/B version of a video segment on demand. As in AB mode, another component further downstream is responsible for creating a unique sequence of A-B video segments in order to encode a payload, e.g. using playlist manipulation or redirection at the edge.

ABRSEIALIGNED: the library incorporates SEI NALUs to the ES AUs and the watermarking pace is set according to the ABR video fragmentation. ContentArmor Embedder shall be placed further downstream (typically at the edge) to embed the desired payload in the video.

For more information on the integration of watermarking technologies in OTT environments, the interested reader is redirected to [1].

1.2 Sequence Flow

The sequence flow of ContentArmor Live Profiler library consists of five (5) main steps.

1. General initialization: the application invokes the dedicated function of the ES/TS API to initialize the internals of ContentArmor Live Profiler library e.g. the underlying video pipeline framework, the software protection watchdog, the logger, etc.
2. Profiler instance creation: the application shall create a dedicated ContentArmor Live Profiler instance for each video source to be processed. It is possible to create several instances in the same process to profile several video sources in parallel.

3. Video source setup: for each created instance of ContentArmor Live Profiler, the application shall declare the individual input video elementary streams that are composing the video source associated to the instance. Depending on the operation mode of the library, one or several video components can be declared for a single instance. Moreover, the application shall declare which video component shall be used as reference for resynchronization purposes at detection.

4. Profiling: once all the video components of a video source have been enlisted in an instance of ContentArmor Live Profiler, the application can initiate profiling operations by feeding the library with video content on a regular basis. Depending on the ES/TS interface, the library is fed in PUSH or PULL mode. Once operations have started, ContentArmor Live Profiler library issues a number of notifications to the main application through the use of callback functions.

5. Profiler instance termination: the application can terminate a running instance of ContentArmor Live Profiler at any time by invoking the dedicated function of the ES/TS API.

1.3 Other Interfaces

1.3.1 Video I/O

ContentArmor Live Profiler library can operate with video feeds that adhere to ContentArmor Encoding Guidelines [2]. Please liaise with your ContentArmor representative if you do not have access to this document.

1.3.1.1 SEI preprovisioning

ContentArmor Live Profiler library does not have TS repacketization capabilities. As a result, when ContentArmor Live Profiler library is integrated using the TS API and when the operating mode is expected to produce SEI NALUs, the video feeds input to the library shall feature pre-provisioned SEI NALUs that will be filled with meaningful data by the library.

ContentArmor is using `user_data_unregistered` SEI NALUs that are placed before the first slice NALU of a non-referenced AU. As such, pre-provisioned SEI NALUs shall adhere to the following template `[codec_sei_header][sei_payload_size][uuid_sei][tiny_sei][sei_filler]` where:

- `[codec_sei_header]` is set to `0x0000 0106 05` in MPEG AVC or `0x0000 014E 0105` in MPEG HEVC;
- `[sei_payload_size]` is set according to the bitrate of the video feed using MPEG size encoding convention;
- `[uuid_sei]` is set to `0x436F 6E74 656E 7441 726D 6F72 5F56 574D`;
- `[tiny_sei]` is set to `0xF782 FD49 1405 FFFF 4F4F`;
- `[sei_filler]` contains `0xFFFF` filling data to match the `[sei_payload_size]`.

ContentArmor recommends to scale the length of the preprovisioned SEIs with the bitrate with a minimum of 48 bytes and a maximum of 512 bytes. As a reference, the following formula: `sei_payload_size = round(max(48,min(512,770*log10(bitrate_kbps)-2032)))` yields 48-bytes long SEIs until 500 kbps and 512-bytes long SEIs beyond 2 Mbps.

When using the ES interface, the integrator may delegate the insertion of the SEI NALUs in the video feed to ContentArmor Live Profiler library. In this case, the induced overhead shall be accurately accounted for by the en-/trans-coder to manage the Video Buffering Verifier (VBV).

1.3.1.2 ABR fragmentation signaling

For OTT ABR delivery, ContentArmor Live Profiler needs information about the video fragmentation that will be later enforced by the video delivery protocol e.g. HLS, DASH, other. More specifically, the application needs to signal all delivery chunk boundaries as well as aligned delivery chunk boundaries. The signaling of these boundaries is explicit in the ES API by properly setting some binary flags when pushing an ES AU for processing. In contrast, in the TS API, such signaling may be implicit e.g. using ATS EBP markers at the TS level [3,4]. In both cases, the watermark embedding pace is set by design by the aligned chunk boundaries possibly complemented by PTS information.

1.3.2 Configuration

ContentArmor Live Profiler library uses a configuration file `config.ini` to set up a number of internal parameters. Those parameters are static and shared by all the concurrent instances of ContentArmor Live Profiler library. Such parameters includes for instance the secret spreading sequence of the client and the operation mode of the library.

1.3.3 Forensic metadata

The library produces Watermark Forensic Metadata (WFM) at regular intervals. This data will be required to perform watermark detection and shall be safely stored and properly indexed. The location where the library places WFM on disk is specified in the configuration file `config.ini`.

```
[dbpush]
dboutputdir="/tmp/"  // Metadata will be placed in subdirectories of this folder.
                     // (one per instance of ContentArmor Live Profiler)
```

In some particular operation modes, it is possible to move the generation of metadata out of ContentArmor Live Profiler with an additional system component placed further downstream. Please contact your ContentArmor representative if you want to enable this feature.

1.3.4 Log mechanism

ContentArmor Live Profiler library logs a number of information. Log messages are sent by default to the standard output interface (`stdout`) and may be redirected by the application to manage their logs. Destination of log messages is set using configuration file config.ini. ContentArmor Live Profiler library uses five level of logs (`error`, `warning`, `info`, `debug`, `specific`). The log level is set using the configuration file `config.ini`.

```
[log]
level="info"  // ContentArmor Live Profiler will provide ERROR, WARNING and INFO log messages
output="/var/log/contentarmor/tsprofiler.log" // Full path to the output log file
```

`[log] output` option can be templated with "%d" string that will be replaced by a timestamp corresponding to the CA_Profiler_init() call date. For example, if CA_Profiler_init() is called the 91th day of year 2021 at 14h37m5s and `[log] output` path is specified as "/var/log/contentarmor/tsprofiler_%d.log" then the log messages will routed to the /var/log/contentarmor/tsprofiler_21091143705.log path.

Warning if the library does not have r/w access to the `[log] output` path, it will run in silent mode without posting any logs message.

The library can create missing directories in the path, provided it runs with the necessary permissions. The "%d" templated string can be used in the parent directories as well as in the file name. Although this is not very useful, several "%d" in the path is also allowed. `[log] output` allows both absolute path orpath relative to the current working directory of the application. If the path is specified as only a file name, then the directory will be the current working directory of the application.

`[log] output`, instead of pointing to a regular file path, can point to a Linux named pipe path. In this case, the named pipe should be created before the CA_Profiler_init() call.

1.3.5 Health monitoring

ContentArmor Live Profiler provides statistics and alerts through a callback mechanism. Monitoring information is emitted on a regular basis. The duration of this aggregation window can be modified in the configuration file `config.ini`.

```
[camonitor]
aggwin=300000  // The aggregation window to report monitoring message is 5 minutes long.
```

1.4 Software Protection

ContentArmor Live Profiler library is protected by a license with an expiration date. Once the license has expired, the Profiler operates in passthrough mode and the watermarking functionality is therefore deactivated. The application can probe the library for the remaining license duration in order to raise relevant alarms and, possibly, trigger the license update procedure. Due to the different software protection mechanisms, linking the library is performed with a mock library at compilation time. The library shall then be substituted by the real implementation to run the application.

1.5 Support

For enquiries during the integration of ContentArmor Live Profiler library, please send emails to Support at support@contentarmor.net

1.6 References

[1] *Forensic Watermarking Implementation Considerations for Streaming Media*, Streaming Video Alliance, July 2018.

[2] *Video Watermark v. 3.0, Encoding Guidelines*, ContentArmor, February 2019

[3] *OpenCable Specifications, Adaptive Transport Stream Specification*, CableLabs, February 2014

[4] *OpenCable Specifications, Encoder Boundary Points Specification*, CableLabs, January 2013

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

3 File Documentation

3.1 CA_profilerAPI.h File Reference

Generic declarations for ContentArmor Live Profiler.

Data Structures

- struct t_CA_Rational

  Data structure used for rational numbers. More...
- struct t_CA_ProfilerConfig

  Data structure for Profiler configuration. More...

Macros

- #define CA_EXPORT __attribute__ ((visibility ("default")))

Typedefs

- typedef void(∗ t_CA_ProfilerMonitoringCallBack)(const char ∗p_jsonString)

  Monitoring callback function to be implemented by the application.
- typedef void(∗ t_CA_ProfilerPushWFMCallBack)(unsigned int p_profilerId, const char ∗p_filePath)

  WFM push callback function to be implemented by the application.

Enumerations

- enum t_CA_VideoCodec { CA_VIDEO_CODEC_AUTO = 0, CA_VIDEO_CODEC_UNKNOWN = 0, CA_VIDEO_CODEC_AVC, CA_VIDEO_CODEC_HEVC }

  List of supported video codecs.
- enum t_CA_ProfilerStatus {
  CA_SUCCESS = 0, CA_INVALID_LICENSE, CA_LICENSE_EXPIRED, CA_MEMORY_ERROR,
  CA_INPUT_QUEUE_FULL, CA_INVALID_PROFILERID, CA_INVALID_STREAMID, CA_INVALID_DURATION,
  CA_INVALID_ARGUMENT, CA_ERROR_NOSTREAM, CA_ERROR_PROFILER_RUNNING, CA_LIB_NOT_INITIALIZED,
  CA_INTERNAL_ERROR }

  Enum of event error codes and function return codes.

Functions

- CA_EXPORT t_CA_ProfilerStatus CA_Profiler_init (const char *p_licensePath, t_CA_ProfilerMonitoringCallBack p_monitoringCallback)

  This function initializes the underlying video pipeline framework of ContentArmor Live Profiler library.

- CA_EXPORT t_CA_ProfilerStatus CA_Profiler_deinit ()

  This function de-initializes the underlying video pipeline framework of ContentArmor Live Profiler library.

- CA_EXPORT t_CA_ProfilerStatus CA_Profiler_getLicenseRemainingTime (unsigned int *p_remainingTime)

  This function is used to obtain the remaining license time of ContentArmor Live Profiler library.

- CA_EXPORT t_CA_ProfilerStatus CA_Profiler_updateLicense (const char *p_licensePath)

  This function is used to update the license of ContentArmor Live Profiler library.

- CA_EXPORT const char * CA_Profiler_getVersion ()

  This function is used to obtain the version of ContentArmor Live Profiler library.

- CA_EXPORT unsigned int CA_Profiler_getNbOutputByStream ()

  This function is used to obtain the number of output streams for each input stream.

- CA_EXPORT t_CA_ProfilerStatus CA_Profiler_getProfilerCustomData (unsigned int p_profilerId, const void **p_customData)

  This function returns the private data that the application may have attached to a given instance of ContentArmor Live Profiler.

- CA_EXPORT t_CA_ProfilerStatus CA_Profiler_getStreamCustomData (unsigned int p_profilerId, unsigned int p_streamId, const void **p_customData)

  This function returns private application data associated to an input video source of a running instance of ContentArmor Live Profiler.

3.1.1 Data Structure Documentation

3.1.1.1 struct t_CA_Rational

This data structure is used to represent rational numbers by indicating their numerator and denominator. It is typically used to specify the fps of a video source.

Data Fields

int	num	Numerator of the rational number.
int	denom	Denominator of the rational number.

3.1.1.2 struct t_CA_ProfilerConfig

This data structure is used to specify several configuration parameters when crating a new Profiler instance. It includes watermark pace making information as well as the frequency of forensic metadata push.

Data Fields

unsigned int	m_payloadSize	Length in bits of the watermark payload. If set to zero, the watermark payload is considered to be infinite.
unsigned int	m_chunksPerBit	In OTT distribution, this number indicated how many segments shall be used to encode a single watermark payload bit. For, instance, if it is set to 1, it indicates that the index of the watermark payload bit shall be updated each and every segment. When it is set to 0, it signals that the watermark pace making logic is based on the number of marks per segment rather than the number of segments.
unsigned int	m_durationPer-Bit	In OTT distribution, when this value is not set to zero, it indicates that the watermark payload bit assigned to be embedded in a given segment shall depend on the PTS information of the ATS EBP marker. For instance, the watermark pace making logic may assign a watermark bit index using the following formula: `WM_bit_idx = floor(segment_pts_in_-ms / m_durationPerBit) % m_payloadSize`. As such, this configuration parameter can be viewed as the duration of a watermark bit in milliseconds.
unsigned int	m_nbMinMarks	This configuration parameter is ignored unless `m_chunksPerBit` is set to 0. In that case, the watermark pace making logic guarantees that the same watermark bit keeps being embedded as long as there is not at least `m_nbMinMarks` along the worst case OTT delivery path.
unsigned int	m_wfmSegment-Duration	Duration in seconds between two pushes of WFM. This configuration parameter specifies how frequently ContentArmor Live Profiler library shall invoke the callback function `t_CA_ProfilerPushWFMCall-Back()` to flush WFM tarballs.

3.1.2 Macro Definition Documentation

3.1.2.1 #define CA_EXPORT __attribute__ ((visibility ("default")))

3.1.3 Typedef Documentation

3.1.3.1 typedef void(* t_CA_ProfilerMonitoringCallBack)(const char *p_jsonString)

ContentArmor Live Profiler library issues at the end of every aggregation window monitoring information such as statistics and alerts. On such occasions, this callback function is invoked by the library to notify the application that new monitoring information is available. It is then up to the application to forward this monitoring information to the supervision layer e.g. using an SNMP agent. The duration of the aggregation window, and thereby the frequency of the monitoring messsages can be configured in the `config.ini` file.

ContentArmor Live Profiler emits three classes of monitoring messages: `counter`, `stat` and `gauge`. Monitoring information is formatted as a JSON string containing the following keys:

JSON key	Class	Description
`time`	all	Timestamp of the last measurement taken into account in the aggregation window using the following format: `DD/MM/YYYY HH:mm:ss.SSS`, where: • `DD` is the day in month (01-31) using two digits, • `MM` is the month in year (01-12) using two digits, • `YYYY` is the year using four digits, • `HH` is the hour in day (00-23) using two digits, • `mm` is the minute in hour (00-59) using two digits, • `ss` is the seconds in minute (00-59) using two digits, • `SSS` is the milliseconds in second (000-999) using three digits.
`type`	all	`metric` only for the time being. Other types may be introduced in the future.
`profiler`	all	[optional] Identifies the Profiler instance associated to the monitoring message.
`stream`	all	[optional] Identifies the stream associated to the monitoring message.
`tag`	all	Identifies the metric associated to the monitoring message. ContentArmor Live Profiler currently provides monitoring information for the metrics identified by the following tags: • `KFDrop` (`counter`): number of anchor frames dropped because there was no room in the fingerprinting queue; • `NoWemListAvailable` (`counter`): number of times the video processing pipeline has been delayed due to a shortage of WEM; • `NoWmiListAvailable` (`counter`): number of times the video processing pipeline has been delayed du to a shortage of WMI; • `PPSMiss` (`counter`): number of times watermark embedding changes had to be discarded because there was no pre-provisioned SEI;

count	all	Number of measurements received during the aggregation window. If no measurement has been received for a metric during the aggregation window, no monitoring message is escalated emitted by ContentArmor Live Profiler.
max	stat	Maximum value of the measurements received during the aggregation window.
min	stat	Minimum value of the measurements received during the aggregation window.
sum	stat	Sum of the measurements received during the aggregation window.
avg	stat	Average value of the measurements received during the aggregation window.
val	gauge	Last measurement received during the aggregation window.

Examples:

- Counter: {"time":"16/12/2019 16:15:36.148","type":"metric","profiler"-:"1","stream":"2","tag":"NoWmiListAvailable","count":"44"}
- Stat: {"time":"16/12/2019 16:15:46.773","type":"metric","profiler"-:"1","stream":"1","tag":"Latency","count":"1489","max":"458","min"-:"0","sum":"115357","avg":"77"}
- Gauge: {"time":"17/12/2019 11:42:28.019","type":"metric","tag":"Lic-RemDay","count":"1","val":"24"}

Parameters

in	*p_json-String*	JSON-formatted string containing monitoring information.

3.1.3.2 typedef void(∗ t_CA_ProfilerPushWFMCallBack)(unsigned int p_profilerId, const char ∗p_filePath)

ContentArmor Live Profiler library generates Watermark Forensic Metadata (WFM) while processing video content. This metadata is flushed at regular intervals as a `tar.gz` tarball. On such occasions, this callback function is invoked by the library to notify the application to safely store this metadata archive and to index it in a reliable database. This metadata will indeed be required to operate detection at a later stage. The location where the WFM tarball are placed as well as the frequency at which they are created are configurable through the configuration file (`config.ini`) of ContentArmor Live Profiler library. The name of the WFM tarball name is built from its starting date and takes the form of `YYDDDHHMMSSmmm_i.tar.gz`, where :

- `YY` is the number of year since 2000 using two digits (00-99);

- DDD is the number of days + 1 since January 1st using three digits (001-366);
- HH is the number of hours after midnight using two digits (00-23);
- MM is the number of minutes after the hour using two digits (00-59);
- SS is the number of seconds after the minute using two digits (00-59, and possibly 60 for leap seconds)
- mmm is the number of milliseconds after the second using 3 digits (000-999)
- i is the index of the WFM segment encoded using 32 bits (variable number of digits).

The timestamp YYDDDHHMMSSmmm is based on the UTC wall-clock time of the local host running ContentArmor profiling library.

The index i of the WFM segment is intended to be incremental by construction. For BROADCAST operating mode, it is equal to the DTS of the first AU comprised in the WFM segment divided by the duration of a WFM segment (in nanoseconds). For ABR operating modes, it is equal to the timestamp of the first aligned chunk comprised in the WFM segment divided by the duration of a WFM segment (in nanoseconds). If the option useebpacquisitiontime is set to true in the profiler section of the configuration file, the timestamp of a chunk is the value of the field EBP_acquisition_time of ATS EBP signaling. Otherwise, it is the DTS of the first AU of the chunk.

Parameters

in	*p_profilerId*	Positive integer identifying an existing instance of ContentArmor Live Profiler. This value has been typically obtained by previously calling an instance creation function.
in	*p_filePath*	NULL-terminated string containing the absolute path of a .tar.gz file containing Watermark Forensic Metadata.

Note

The management of the WFM database is critical to guarantee E2E operations of ContentArmor Video Watermarking system. System designs shall therefore be discussed and approved by a ContentArmor representative.
In some operation modes, WFM is generated further downstream in the distribution pipeline. In such cases, this callback is never invoked by ContentArmor Live Profiler library.

3.1.4 Enumeration Type Documentation

3.1.4.1 enum t_CA_VideoCodec

List of supported video codecs

Note

ContentArmor Live Profiler only supports MPEG AVC and HEVC video streams.

Enumerator

CA_VIDEO_CODEC_AUTO Codec to be retrieved from the PMT

CA_VIDEO_CODEC_UNKNOWN Other codecs

CA_VIDEO_CODEC_AVC MPEG AVC video stream

CA_VIDEO_CODEC_HEVC MPEG HEVC video stream

3.1.4.2 enum t_CA_ProfilerStatus

Enum of event error codes and function return codes

Enumerator

CA_SUCCESS The requested action has been performed successfully.

CA_INVALID_LICENSE The license file provided in input is invalid.

CA_LICENSE_EXPIRED The license file provided in input has expired.

CA_MEMORY_ERROR An error occured because memory could not be allocated.

CA_INPUT_QUEUE_FULL The requested action could not be performed because the processing queue is currently full.

CA_INVALID_PROFILERID The requested action could not be performed because there is no ContentArmor Live Profiler instance with the ID provided in input.

CA_INVALID_STREAMID The requested action could not be performed because the ContentArmor Live Profiler instance does not have a video source associated to the stream ID provided in input.

CA_INVALID_DURATION The requested action could not be performed because the frame duration provided in input is not compliant with the ones of the previously declared video sources.

CA_INVALID_ARGUMENT The requested action could not be performed because one of the input parameters is invalid.

CA_ERROR_NOSTREAM An issue occurred because no video source is currently associated to the instance of ContentArmor Live Profiler.

CA_ERROR_PROFILER_RUNNING Some action could not be performed because the instance of ContentArmor Live Profiler is currrently running.

CA_LIB_NOT_INITIALIZED The requested action could not be performed because the library has not been initialized.

CA_INTERNAL_ERROR An internal error occured.

3.1.5 Function Documentation

3.1.5.1 CA_EXPORT t_CA_ProfilerStatus CA_Profiler_init (const char * *p_licensePath*, t_CA_ProfilerMonitoringCallBack *p_monitoringCallback*)

ContentArmor Live Profiler library requires to initialize some internals of its video processing pipeline prior to launching any operation. This function requires the full path to a valid ContentArmor Live Profiler license file to terminate successfully. ContentArmor Live Profiler library is protected by a license with an expiration date. If ContentArmor Live Profiler library cannot be successfully initialized due to an issue related to the license, it operates in passthrough mode and the watermarking functionality is therefore deactivated.

Parameters

in	*p_licensePath*	Full path to a ContentArmor Live Profiler license file.
in	*p_monitoring-Callback*	Callback function used by ContentArmor Live Profiler to notify the application that some monitoring data (statistic or alert) is available. If set to NULL, no monitoring information is provided by ContentArmor Live Profiler.

Return values

CA_SUCCESS	ContentArmor Live Profiler library has been successfully initialized.
CA_INVALID_ARGUMENT	ContentArmor Live Profiler library has not been initialized properly because the path to the provided license file is invalid. Either no path has been provided, or the file does not exist, or the library does not have r/w access to this file.
CA_LICENSE_EXPIRED	ContentArmor Live Profiler library has not been initialized properly because the licensing period declared in the provided license file is over.
CA_INVALID_LICENSE	ContentArmor Live Profiler library has not been initialized properly because the provided license file is not valid.
CA_INTERNAL_ERROR	ContentArmor Live Profiler library has not been initialized properly because the underlying video processing framework initialization failed.

3.1.5.2 CA_EXPORT t_CA_ProfilerStatus CA_Profiler_deinit ( )

When application has no running instance of ContentArmor Live Profiler and it no longer needs to open a new one, this function can be called to release ContentArmor Live Profiler Library resources.

Return values

CA_SUCCESS	ContentArmor Live Profiler library has been successfully de-initialized.
CA_ERROR_PROFILER_-RUNNING	The library has not been de-initialized because some instances of ContentArmor Live Profiler are still running.
CA_LIB_NOT_INITIALIZED	The requested action could not be performed because the library has not been previously initialized.

3.1.5.3 CA_EXPORT t_CA_ProfilerStatus CA_Profiler_getLicenseRemainingTime (unsigned int * *p_remainingTime*)

ContentArmor Live Profiler library is protected by a license with an expiration date. Once the license has expired, the Profiler operates in passthrough mode and the watermarking functionality is therefore deactivated. This function is intended to provide means for the application to probe the library for the remaining license duration on a regular basis and thus be alerted when the license expiration date is approaching. The Operator can then take the necessary steps to update the license file.

Parameters

`out`	*p_remaining-Time*	Pointer to memory where the library shall store the remaining number of seconds before the license expires.

Return values

CA_SUCCESS	The license is valid and there is still remaining time.
CA_LIB_NOT_INITIALIZED	The requested action could not be performed because the library has not been initialized.
CA_LICENSE_EXPIRED	The license is valid but there is no remaining time.
CA_INVALID_LICENSE	The license is not valid.

3.1.5.4 CA_EXPORT t_CA_ProfilerStatus CA_Profiler_updateLicense (const char * *p_licensePath*)

ContentArmor Live Profiler library is protected by a license with an expiration date. Once the license has expired, the Profiler operates in passthrough mode and the watermarking functionality is therefore deactivated. This function provides means to the Operator to update the license file without stopping running instances of ContentArmor Live Profiler library. This function requires the full path to a ContentArmor license file. If the update operation is successfull with a valid license, ContentArmor Live Profiler library operates normally, possibly resuming nominal operations after a period in passthrough mode while the license was invalid/expired (before the update action).

Parameters

in	*p_licensePath*	Full path to a ContentArmor Live Profiler license file. This file is used to update a running instance of ContentArmor Live Profiler library.

Return values

CA_SUCCESS	ContentArmor Live Profiler license has been successfully updated.
CA_LIB_NOT_INITIALIZED	ContentArmor Live Profiler license has not been updated because the initialization process of the libary has not terminated succesfully prior to this call.
CA_INVALID_ARGUMENT	ContentArmor Live Profiler license has not been updated because the path to the provided license file is invalid. Either no path has been provided, or the file does not exist, or the library does not have r/w access to this file.
CA_LICENSE_EXPIRED	ContentArmor Live Profiler license has not been updated because the licensing period declared in the provided license file is over.
CA_INVALID_LICENSE	ContentArmor Live Profiler library has not been updated because the provided license file is not valid.

3.1.5.5 CA_EXPORT const char* CA_Profiler_getVersion ( )

This function is used to obtain the version of ContentArmor Live Profiler library.

Returns

NULL-terminated string containing the versioning information

3.1.5.6 CA_EXPORT unsigned int CA_Profiler_getNbOutputByStream ( )

Depending on the operating mode declared in ContentArmor Live Profiler configuration file, the library may have a different number of output streams per input stream. Typically, there is a one-to-one relationship except in AB mode where there are two output streams per input. This function provides means to the application to know this number.

Returns

Number of output streams for each input stream

3.1.5.7 CA_EXPORT t_CA_ProfilerStatus CA_Profiler_getProfilerCustomData (unsigned int *p_profilerId,* const void ** *p_customData*)

The application has the opportunity to attach custom data to an instance of ContentArmor Live Profiler using the parameter `p_customData` of an instance creation function. This function provides means to the application to recover this custom data. This may be useful to implement callback functions when several instances are running concurrently.

Parameters

in	*p_profilerId*	Positive integer identifying an existing instance of ContentArmor Live Profiler. This value has been typically obtained by previously calling an instance creation function.

out	*p_customData*	Pointer to the custom data attached to the instance of ContentArmor Live Profiler ID p_profilerId. It is NULL if no custom data has been provided at the creation time of the instance.

Return values

CA_SUCCESS	ContentArmor Live Profiler successfully returned the requested custom data.
CA_INVALID_PROFILERID	ContentArmor Live Profiler failed to return the requested custom data because there is no ContentArmor Live Profiler instance with the ID value p_profilerId.

3.1.5.8 CA_EXPORT t_CA_ProfilerStatus CA_Profiler_getStreamCustomData (unsigned int *p_profilerId,* unsigned int *p_streamId,* const void ** *p_customData*)

The application has the opportunity to attach custom data to each video source of a running instance of ContentArmor Live Profiler using the parameter p_customData of a video source declaration function. This function provides means to the application to recover this custom data.

Parameters

in	*p_profilerId*	Positive integer identifying an existing instance of ContentArmor Live Profiler. This value has been typically obtained by previously calling an instance creation function.
in	*p_streamId*	Integer value that identifies which video source of the ContentArmor Live Profiler instance shall be considered for this function call. This value has been typically obtained by previously calling a video source declaration function.
out	*p_customData*	Pointer to the custom data attached to the input video stream p_streamId of ContentArmor Live Profiler instance p_profilerId.

Return values

CA_SUCCESS	ContentArmor Live Profiler successfully returned the requested custom data.
CA_INVALID_PROFILERID	ContentArmor Live Profiler failed to return the requested custom data because there is no ContentArmor Live Profiler instance with the ID value p_profilerId.
CA_INVALID_STREAMID	ContentArmor Live Profiler failed to return the requested custom data because the instance of ContentArmor Live Profiler does not feature any video source with the stream ID value provided in input.

3.2 CA_profilerTsAPI.h File Reference

TS specific extension for ContentArmor Live Profiler Application Programming Interface.

Macros

- #define CA_TS_PACKET_SIZE (188)

  Size in bytes of a TP.
- #define CA_PID_AUTO 0x1FFF

  Value to be used when the PID must be retrieved from the stream.

Typedefs

- typedef unsigned int(∗ t_CA_ProfilerInputTPsCallBack)(unsigned int p_profilerId, unsigned int p_streamId, unsigned char ∗p_buffer, unsigned int p_numPackets)

  Callback function to be implemented by the application to provide input TPs to ContentArmor Live Profiler library.
- typedef void(∗ t_CA_ProfilerOutputTPsCallBack)(unsigned int p_profilerId, unsigned int p_streamId, unsigned int p_outputIndex, const unsigned char ∗p_buffer, unsigned int p_numPackets)

  Callback function to be implemented by the application to flush TPs output by ContentArmor Live Profiler library.

Functions

- CA_EXPORT t_CA_ProfilerStatus CA_ProfilerTS_open (t_CA_ProfilerConfig p_config, const void ∗p_customData, t_CA_ProfilerInputTPsCallBack p_inputTPsCallBack, t_CA_ProfilerOutputTPsCallBack p_outputTPsCallBack, t_CA_ProfilerPushWFMCallBack p_pushWFMCallBack, unsigned int ∗p_profilerId)

  This function creates an instance of a ContentArmor Live Profiler.
- CA_EXPORT t_CA_ProfilerStatus CA_ProfilerTS_addStream (unsigned int p_profilerId, const void ∗p_customData, bool p_isReferenceStream, unsigned int p_videoPid, unsigned int p_bitrate, t_CA_VideoCodec p_codec, t_CA_Rational p_frameRate, unsigned int p_maxAuRxTime, unsigned int ∗p_streamId)

  This function adds an input video stream to be profiled by a ContentArmor Live Profiler.
- CA_EXPORT t_CA_ProfilerStatus CA_ProfilerTS_start (unsigned int p_profilerId)

  This function starts processing all input video streams to be profiled by an existing instance of ContentArmor Live Profiler.
- CA_EXPORT t_CA_ProfilerStatus CA_ProfilerTS_close (unsigned int p_profilerId)

  This function terminates an existing ContentArmor Live Profiler instance.

3.2.1 Detailed Description

When ContentArmor Live Profiler API is used at the transport stream (TS) level, the processing pipeline operates Transport Packet (TP) by TP. Once an instance has been created and configured, the library is pulling on a regular basis lines of input TPs which the application is expected to turn in rapidly. The library also uses callback functions to notify the application (i) to request memory allocation, (ii) to flush TPs further downstream, and (iii) to notify that forensic metadata has been created and shall be securely stored and properly indexed in a database.

Note

It is on ContentArmor's roadmap to switch to a PUSH input at some point in time. One of the objectives is to be VBR compliant.

3.2.2 Macro Definition Documentation

3.2.2.1 #define CA_TS_PACKET_SIZE (188)

Size in bytes of a TP

3.2.2.2 #define CA_PID_AUTO 0x1FFF

Value to be used when the PID must be retrieved from the stream

3.2.3 Typedef Documentation

3.2.3.1 typedef unsigned int(∗ t_CA_ProfilerInputTPsCallBack)(unsigned int p_profilerId, unsigned int p_streamId, unsigned char ∗p_buffer, unsigned int p_numPackets)

When ContentArmor Live Profiler API is used at the TS level, the library operates in PULL mode and is responsible for requesting TPs to process to the application. To do so, it invokes this callback function to notify the application to provide additional TPs to process. The application is expected to return these TPs as fast as possible.

Parameters

in	*p_profilerId*	Positive integer identifying an existing instance of ContentArmor Live Profiler. This value has been typically obtained by previously calling CA_Profiler-TS_open().
in	*p_streamId*	Integer value that identifies which input video source of the ContentArmor Live Profiler instance shall be considered for this function call.
out	*p_buffer*	Pointer to a buffer that needs to be filled with TPs coming from the input video source p_streamId. Memory allocation and release of this buffer is managed inside ContentArmor Live Profiler library.
in	*p_numPackets*	Number of TPs to copy in the buffer p_buffer.

Returns

The number of TPs effectively copied by the application in the buffer p_buffer.

3.2.3.2 typedef void(∗ t_CA_ProfilerOutputTPsCallBack)(unsigned int p_profilerId, unsigned int p_streamId, unsigned int p_outputIndex, const unsigned char ∗p_buffer, unsigned int p_numPackets)

ContentArmor Live Profiler library invokes this callback function when it finalizes processing an ES AU and wants to flush the corresponding TPs out of its processing pipeline. The library expects a fast turn-around time of the function as it is on the critical path of the processing pipeline. The application shall assume that the memory pointed to by p_buffer will no longer be available after this function exits.

Parameters

in	*p_profilerId*	Positive integer identifying an existing instance of ContentArmor Live Profiler. This value has been typically obtained by previously calling CA_Profiler-TS_open().
in	*p_streamId*	Integer value that identifies which input video source of the ContentArmor Live Profiler instance shall be considered for this function call.
in	*p_outputIndex*	Integer value that signals which output stream is being flushed. In mode AB, output index 0 (resp. 1) is associated to the prewatermarked version A (resp. B). In all other modes, there is a single ouput stream per input stream and the output index is set to 0.
out	*p_buffer*	Pointer to a buffer that contains the output TPs for the input video source p-_streamId that need to be pushed further downstream by the application. Memory allocation and release of this buffer is managed inside ContentArmor Live Profiler library.
in	*p_numPackets*	Number of TPs in p_buffer that need to be flushed.

3.2.4 Function Documentation

3.2.4.1 CA_EXPORT t_CA_ProfilerStatus CA_ProfilerTS_open (t_CA_ProfilerConfig *p_config,* const void ∗ *p_customData,* t_CA_ProfilerInputTPsCallBack *p_inputTPsCallBack,* t_CA_ProfilerOutputTPsCallBack *p_outputTPsCallBack,* t_CA_ProfilerPushWFMCallBack *p_pushWFMCallBack,* unsigned int ∗ *p_profilerId*)

An instance of ContentArmor Live Profiler needs to be created before starting processing one or several video streams (in case of OTT ABR delivery). To operate, an instance needs three callback functions to manage the inputs and outputs of the library: p_inputTPsCallBack(), p_outputTPsCallBack(), and p_pushW-FMCallBack(). The application can also attach its own custom data to an instance of ContentArmor Live Profiler if needed. If successful, the function returns a handle to the newly created instance.

Parameters

in	*p_config*	Data structure used to set the per instance configuration parameters.
in	*p_customData*	Pointer to a memory location where opaque data is stored that the application may attach to an instance of ContentArmor Live Profiler to manage its internals, e.g. the implementation of the callback function. It may be set to NULL if the application does not need it.
in	*p_inputTPsCallBack*	Callback function used by an instance of ContentArmor Live Profiler to request input TPs to the application.
in	*p_outputTPsCallBack*	Callback function used by an instance of ContentArmor Live Profiler to flush output TPs out of the processing pipeline.
in	*p_pushWFMCallBack*	Callback function used by ContentArmor Live Profiler to notify the application that some Watermark Forensic Metadata has been produced and shall be safely stored and indexed in a database.
out	*p_profilerId*	Pointer to a memory location where the integer value that identifies the instance of ContentArmor Live Profiler created with this function call is stored. The returned value of p_profilerId is typically incremented after each call to CA_ProfilerTS_open(). In case of error, it is set to NULL.

Return values

CA_SUCCESS	An instance of ContentArmor Live Profiler has been successfully created.
CA_INVALID_ARGUMENT	The creation of a new ContentArmor Live Profiler instance has failed because the provided input arguments are invalid. In general, either the value of p_profilerId is NULL or one of the provided callback functions is NULL. When Watermark Forensic Metadata generation is deported, the callback p_pushWFMCallBack() can be set to NULL.
CA_INVALID_LICENSE	The creation of a new ContentArmor Live Profiler instance has failed because the license file is not valid.
CA_LICENSE_EXPIRED	The creation of a new ContentArmor Live Profiler instance has failed because the licensing period declared in the license file is over.
CA_MEMORY_ERROR	The creation of a new ContentArmor Live Profiler instance has failed because the library failed to allocate memory.
CA_INTERNAL_ERROR	The creation of a new ContentArmor Live Profiler instance has failed because the underlying video processing pipeline could not be created.
CA_LIB_NOT_INITIALIZED	The requested action could not be performed because the library has not been previously initialized.

3.2.4.2 CA_EXPORT t_CA_ProfilerStatus CA_ProfilerTS_addStream (unsigned int *p_profilerId,* const void * *p_customData,* bool *p_isReferenceStream,* unsigned int *p_videoPid,* unsigned int *p_bitrate,* t_CA_VideoCodec *p_codec,* t_CA_Rational *p_frameRate,* unsigned int *p_maxAuRxTime,* unsigned int * *p_streamId*)

Once a ContentArmor Live Profiler instance has been successfully created, video streams have to be incorporated one by one to the video processing pipeline. One of the video streams to be added to an instance of ContenArmor Live Profiler shall be set as the reference to be used for resynchronization purposes. In OTT ABR, this reference stream shall be the video track in the ABR bundle with the highest frame rate, largest resolution, and biggest ES bitrate (in decreasing order of importance). The reference stream is signaled with a dedicated binary flag p_isReferenceStream when adding a video stream to the video processing pipeline.

Parameters

in	*p_profilerId*	Positive integer identifying an existing instance of ContentArmor Live Profiler. This value has been typically obtained by previously calling CA_ProfilerTS_open().

in	*p_customData*	Pointer to custom data that may be attached to the input video stream. provided in input. It shall be set to NULL if no custom data is used.
in	*p_isReferenceStream*	Binary flag used to signal that the added video stream shall be considered as the reference for resynchronization purposes. There shall be a single reference stream per instance of ContentArmor Live Profiler.
in	*p_videoPid*	Positive integer value that gives the PID of the TPs associated to the video source that shall be added to the instance of ContentArmor Live Profiler. This parameter can be set to CA_PID_AUTO if: 1. A Program Mapping Table (PMT) is present in the input transport stream; in that case, ContentArmor Live Profiler selects the first video stream for profiling. 2. There is no PMT but the stream contains a single video stream and only video TPs. In that case, no TP filtering is needed since it has been done prior to feeding ContentArmor Live Profiler library.
in	*p_bitrate*	Positive integer value that gives the TS bitrate in kbps for this input bitstream. This value shall be accurately set as it impacts both (i) the dimensioning of ConentArmor Live Profiler internal TP circular buffer and (ii) the frequency at which the library invokes the callback function p_inputTPsCallBack() to fetch input TPs to process.
in	*p_codec*	Video codec of the input ES video stream. ContentArmor Live Profiler currently supports AVC (CA_VIDEO_CODEC_AVC) and HEVC (CA_VIDEO_CODEC_HEVC) video streams. If the stream contains a PMT, this parameter can be set to CA_VIDEO_CODEC_AUTO to retrieve the codec information from the PMT.
in	*p_frameRate*	Frame rate of the input ES video stream. ContentArmor Live Profiler currently supports ABR bundles where the frame rate of the different tracks is either the same or a multiple of each others.
in	*p_maxAuRxTime*	Positive integer value that gives the maximum reception time of an access unit in microseconds. This value shall be accurately set as it impacts the dimensioning of ContentArmor Live Profiler internal TP circular buffer.
out	*p_streamId*	Pointer to a memory location where the integer value that ContentArmor Live Profiler uses to identify the added video stream is stored. The value of p_streamId is typically incremented after each call to CA_ProfilerTS_addStream().

Return values

CA_SUCCESS	The input video bitstream has been successfully added to the video processing pipeline of this instance of ContentArmor Live Profiler.
CA_INVALID_PROFILERID	The input video bitstream has not been added to the video processing pipeline because there is no ContentArmor Live Profiler instance with the ID value `p_profilerId`.
CA_INVALID_CODEC	The input video bitstream has not been added to the video processing pipeline because the codec provided in input is not supported by ContentArmor Live Profiler.
CA_INVALID_DURATION	The input video bitstream has not been added to the video processing pipeline because the frame duration provided in input is not coherent with the ones declared for other renditions.
CA_INVALID_ARGUMENT	The input video bitstream has not been added to the video processing pipeline because one of the provided input argument is not valid. For instance, • The frame rate provided in input is not coherent with the ones declared for other renditions. • The flag `p_isReferenceStream` is set to `TRUE` whereas another video stream, previously added to this instance of ContentArmor Live Profiler, has been declared as the reference stream.
CA_ERROR_PROFILER_RUNNING	The input video bitstream has not been added to the video processing pipeline because the instance of ContentArmor Live TS Profiler is already started.
CA_MEMORY_ERROR	The input video bitstream has not been added to the video processing pipeline because the library failed to allocate some memory.
CA_INTERNAL_ERROR	The input video bitstream has not been added to the video processing pipeline because an error occured at the video framework level.

3.2.4.3 CA_EXPORT t_CA_ProfilerStatus CA_ProfilerTS_start (unsigned int *p_profilerId*)

Once all video sources have been added to an instance of ContentArmor Live Profiler using the function `CA_ProfilerTS_addStream()`, profiling operations can be started by invoking this function. Once started, the application shall no longer call the function `CA_ProfilerTS_addStream()` to modify the configuration of this instance.

Once an instance of ContentArmor Live Profiler has been started, a thread is initiated for each video stream added to the instance. Each of these threads fetches input TPs for the corresponding video stream, using the callback function `p_inputTPsCallBack()` regularly. ContentArmor library is expecting to be fed with all TPs associated to a video source in transmission order. When processing an ES AU is finalized, the library flushes the corresponding output TPs using the function `p_outputTPsCallBack()`. The threads of the different video streams are terminated when the application calls the function `CA_ProfilerTS_close()`.

In OTT ABR, ContentArmor Live Profiler with a TS interface is relying on ATS EBP markers for watermak pace making. The video segmentation shall be consistent accross the different video sources attached to the instance of ContentArmor Live Profiler.

Parameters

`in`	*p_profilerId*	Positive integer identifying an existing instance of ContentArmor Live Profiler. This value has been typically obtained by previously calling `CA_ProfilerTS_open()`.

Return values

CA_SUCCESS	The instance of ContentArmor Live Profiler has been successfully started.
CA_INVALID_PROFILERID	The instance of ContentArmor Live Profiler has not been started because there is no ContentArmor Live Profiler instance with the ID value `p_profilerId`.
CA_ERROR_NOSTREAM	The instance of ContentArmor Live Profiler has not been started because no video stream has been added to this instance using the function `CA_ProfilerTS_addStream()` prior to this call.
CA_INTERNAL_ERROR	The instance of ContentArmor Live Profiler has not been started because the underlying video processing pipeline could not be started.

3.2.4.4 CA_EXPORT t_CA_ProfilerStatus CA_ProfilerTS_close (unsigned int *p_profilerId*)

When this function is invoked, the running instance of ContentArmor Live Profiler stops accepting new video content in input. The processing of all TPs already present in the different processing queues of the instance is finalized before flushing the result further downstream. Eventually, the instance of ContentArmor Live Profiler is destroyed.

Parameters

`in`	*p_profilerId*	Positive integer identifying an existing instance of ContentArmor Live Profiler. This value has been typically obtained by previously calling an instance creation function.

Return values

CA_SUCCESS	ContentArmor Live Profiler successfully destroyed the specified instance of ContentArmor Live Profiler.
CA_INVALID_PROFILERID	ContentArmor Live Profiler failed to destroy the specified instance of ContentArmor Live Profiler because there is no instance with the ID value `p_profilerId`.

Index