Functions Manual

UVCS Common Engine Library

Functions Manual

Renesas Electronics
www.renesas.com

Rev.1.00 2014.08

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

How to Use This Manual

1. Purpose and Target Readers

This manual is designed to provide the user with an understanding of the user interface of this software.
Target reader is the user which designs the applied system using this software.
For using this manual, it is required following knowledge,

- Knowledge of Moving picture.
- Knowledge of RTOS (Real time operating system).
- Knowledge of each CODEC to use.

2. About using this software

When you use this software, you need to enter into the software license agreement with us.

3. Related Manuals

As a related document, the next manual and document are prepared for using this software.
Please contact Renesas Electronics sales office if necessary.

- UVCS Decode Engine Library User's Manual
- UVCS Encode Engine Library User's Manual

4. About Revision History

Revision history is only the main point that we corrected or added it for the former edition.
It is not the thing which recorded all the revision contents.
Please refer to this manual about a detail.

5. List of Abbreviations

Abbreviation	Long Name
UVCS-CMN	UVCS Common Engine Library
UVCS-DEC	UVCS Decode Engine Library
UVCS-ENC	UVCS Encode Engine Library
DRV-CORE	VCP3 Driver Core Library

6. List of Acronyms

Acronyms	Long Name
UVCS	Unified Video CODEC Server
RTOS	Real-Time Operating System

Table of contents

1. Overview

1.1 Feature

This library (UVCS-CMN) is the one of the library which consists of UVCS. This library is used from UVCS-DEC and UVCS-ENC, and has a function of interrupt handler, scheduler, interface of hardware, and arbiter.

This library has a feature of following.

- User does not need to consider the number of video hardware, etc. (Automatic resource assignment)
- Even if there is a difference in frame-rate etc., two or more contents can be processed in same time. (Scheduler and arbiter)

In normally, each function on this library is called from kernel driver (provided as a sample code). Therefore the user doesn't need to call each function directly.

The relationship between UVCS-DEC, UVCS-ENC, OpenMAX IL, Video-Driver and Video-Hardware is shown in Figure 1.1 and Figure 1.2 .

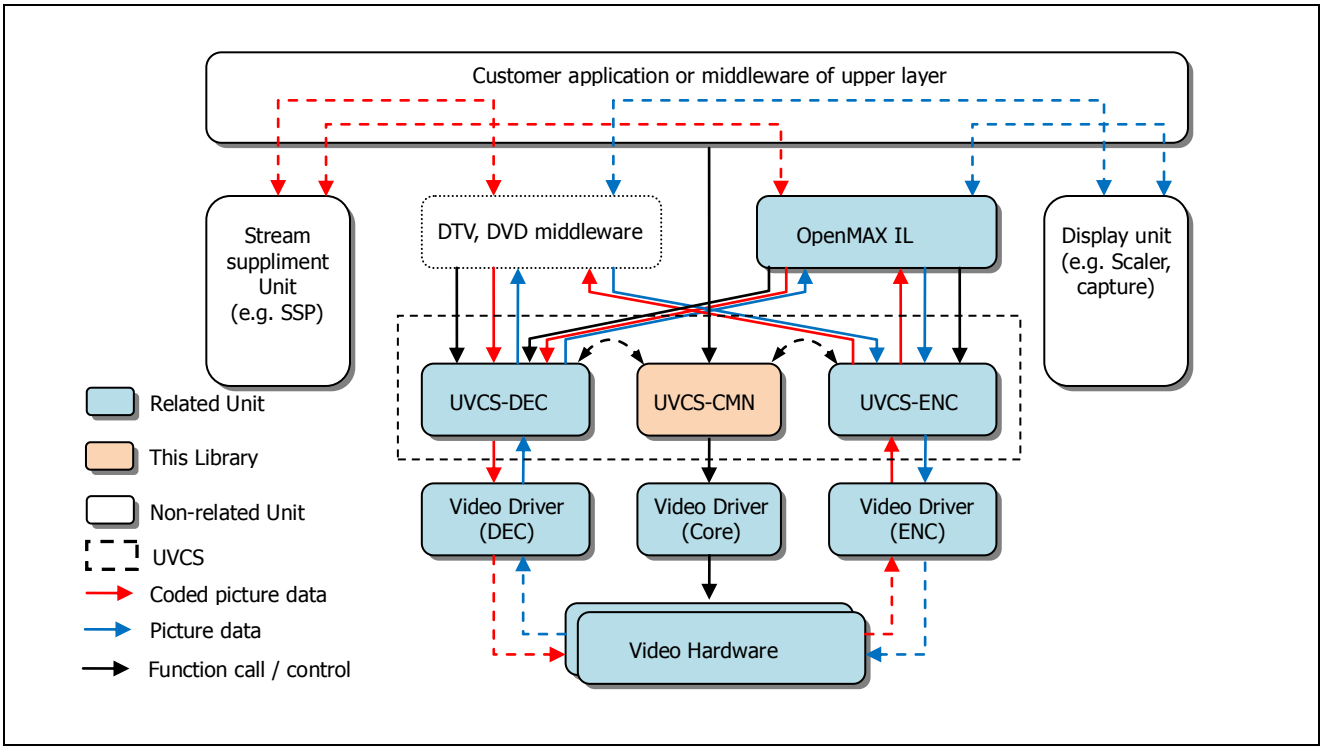

Figure 1.1 software stack

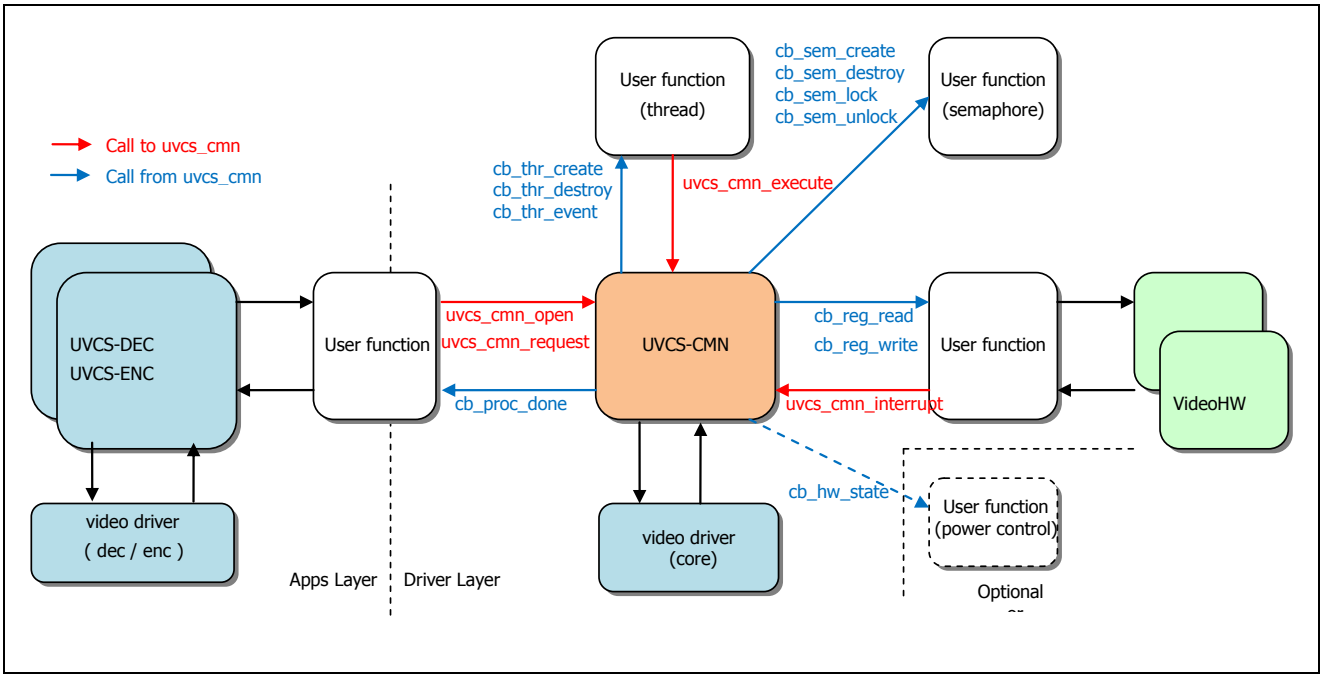

Figure 1.2 Block Diagram of related unit

2. Basic Specification

2.1 Summary Specification

The summary of specification is described in Table 2.1.

Table 2.1 Summary Specification

Items	Description
Decoding Spec.	Please refer the User's Manual of UVCS-DEC.
Encoding Spec.	Please refer the User's Manual of UVCS-ENC.
Programming Languages	C
task / thread	Used.
Semaphore / Mutex	Used.
Maximum handle num	Unconstraint.
Multi-IP support	Yes. Max 2 video hardwares.
Memory area	Please see the section 2.3.
Stack	Maximum 256 byte. (The consumption in Video-Driver and callback are not included)
Interrupt	Used.

2.2 Reserved word

This library uses the following prefixes for avoiding confusion from other software. Using prefixes are described in Table 2.2.

Table 2.2 Prefixes

prefix	Description
UVCS_*	Prefix for UVCS
uvcs_*	

The reserved word of DRV-CORE Library refers the User's Manual of DRV-CORE Library.

2.3 Memory Requirement

The memory area used by this library is described in Table 2.3 and Table 2.4 .

Table 2.3 Memory area

Area	Type	Alignment	Use for
work_mem_0	Cached (virtual)	4	Work memory for this library.
hdl_work_0	Cached (virtual)	4	Work area for every handle.

Table 2.4 Requisite Memory Size

Area	Condition	Memory size
work_mem_0	Fixed size	512 Byte
hdl_work_0	Fixed size (unique area is needed for each handle)	1024 Byte (for each handle)

3. Common Processing

3.1 Initializations

This section shows the initialization of this library. The initialization of this library is needed before using UVCS-DEC and UVCS-ENC, and is performd by calling uvcs_cmn_initialize (4.2.2). The following informations are needed for this function.

- Video hardware base address information
- Number of video hardware to control by UVCS
- Function pointer for callback

3.2 Finite-state machine

This library has three states which are Uninitialized-state (3.2.1), Initialized-state (3.2.2) and Ready-state (3.2.3), and is shown in Figure 3.1.

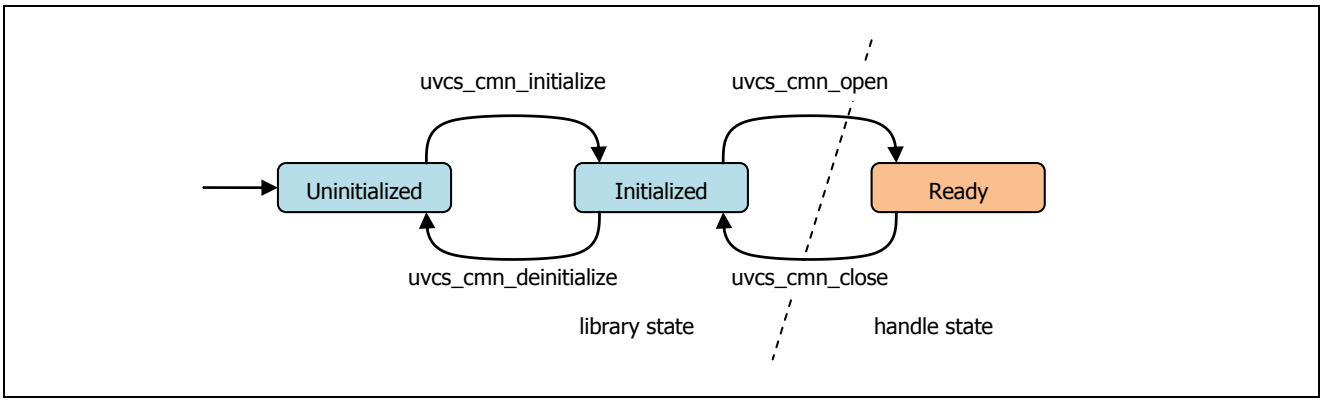

Figure 3.1 Finite-state machine

3.2.1 Uninitialized-state

This state is the default state immediately after booting the system. The resources for working this library have not been allocated. In this state, it is possible to call the uvcs_cmn_initialize (4.2.2). By calling uvcs_cmn_initialize (4.2.2), the state of this library will be changed to initialized-state.

3.2.2 Initialized-state

The state is indicated this library has been initialized. The resources for working this library (such as RTOS task or semaphore) have been allocated. In this state, the user can call uvcs_cmn_deinitialize (4.2.3), uvcs_cmn_interrupt (4.2.1) and uvcs_cmn_open (4.2.4). By calling uvcs_cmn_deinitialize (4.2.3), the state of this library will be changed to uninitialized state. By calling uvcs_cmn_open (4.2.4), the handle will be allocated (into ready state) and the hardware processing becomes to be able to perform.

3.2.3 Ready-state

This state is the handle's state and is possible to start hardware processing. In this state, the user can call uvcs_cmn_request (4.2.6) and uvcs_cmn_close (4.2.5). By calling uvcs_cmn_close (4.2.5), the allocated handle will be released (turns into initialized state). By calling uvcs_cmn_request (4.2.6) and uvcs_cmn_execute (4.2.7), the hardware processing will be started.

3.3 Function Flow

The typical function-flow of this library is shown in Figure 3.2. For more detail of each function, please see the specification of each function. In normally, each function of this library is called by kernel driver (linux cases). Therefore the user does not need to call each function of this library. Please see the sequence diagram of kernel driver (provided as a sample code) and application sample (the User's Manual of UVCS-DEC / UVCS-ENC.).

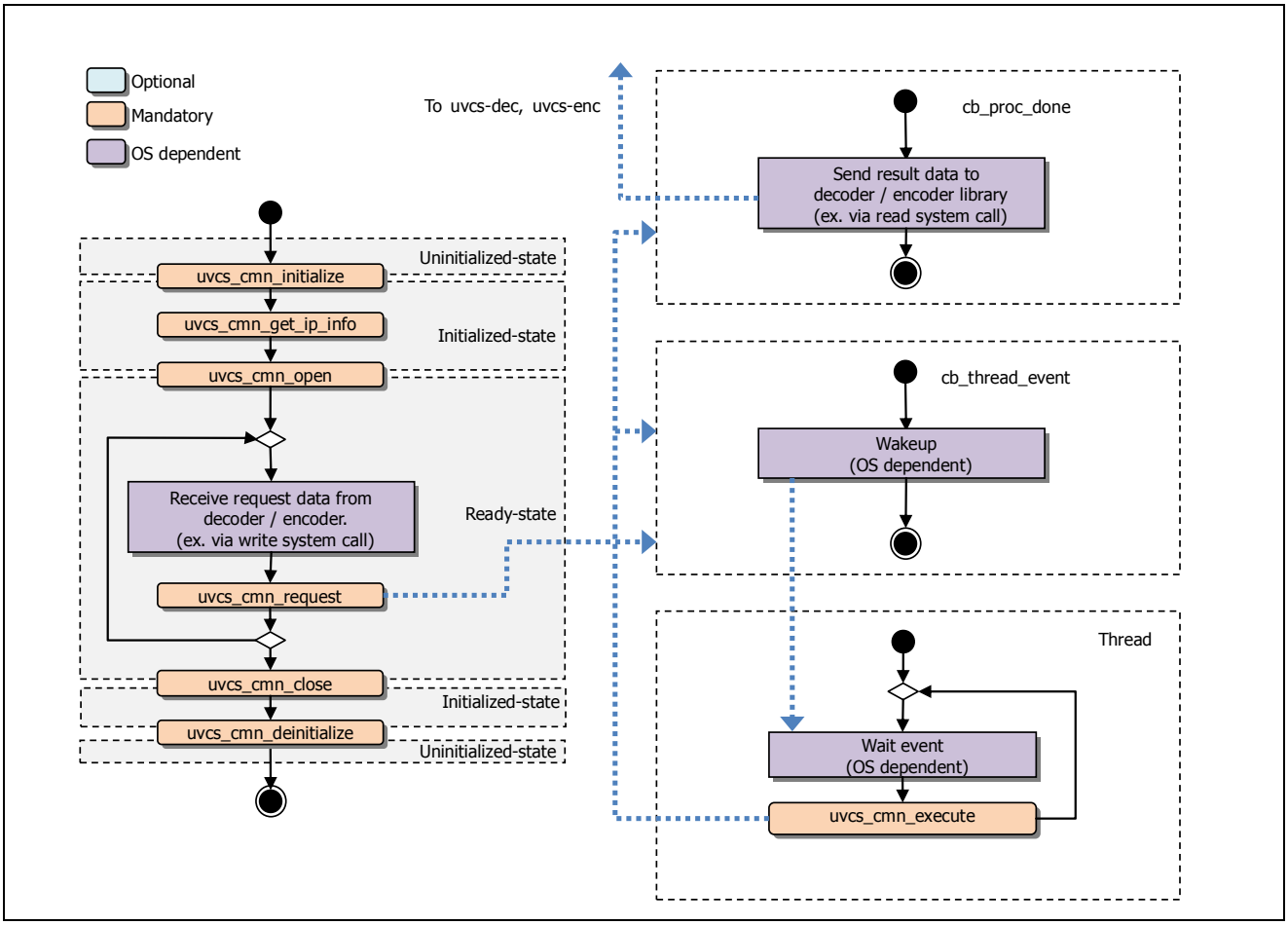

Figure 3.2 UVCS-CMN basic function flow

- First, please call uvcs_cmn_initialize (4.2.2) to perform initialization of this library. For the parameter of this function, please specify the address of video hardware and the number of video hardware to want to handle by this library.
- Next, please call uvcs_cmn_get_ip_info (4.2.10) to read the information of video hardware. This information is used for determining hardware functional difference by the decoder library and encoder library.
- Next, please call uvcs_cmn_open (4.2.4) to open a handle of this library. This handle is used for the communication for decoder library or encoder library. The handle of this library is needed per the context of decoder library or encoder library.
- Next, please call uvcs_cmn_request (4.2.6) to request hardware processing. The request of hardware processing is sended from decoder library or encoder library.
- When the callback function UVCS_CMN_CB_THREAD_EVENT is called, please call uvcs_cmn_execute (4.2.7) from a dedicated thread.
- When the hardware processing is ended, the callback function UVCS_CMN_CB_PROC_DONE will be called. Please send the result information (argument of callback function) to decoder library or encoder library. For detailed information of the communication for decoder library or encoder library, please see the sample code.

4. Functions

4.1 Function List

4.1.1 External Function

The external functions are shown in Table 4.1. And callable timings are shown in 'Executable State' in each function's specification.

Table 4.1 List of External functions

Name	Purpose	Mandatory
uvcs_cmn_interrupt	Notification of video hardware interrupts.	Mandatory
uvcs_cmn_initialize	Initialize this library	Mandatory
uvcs_cmn_deinitialize	De-initialize this library	Mandatory
uvcs_cmn_open	Allocate the handle.	Mandatory
uvcs_cmn_close	Release allocated handle.	Mandatory
uvcs_cmn_request	Requests hardware decoding/encoding.	Mandatory
uvcs_cmn_execute	Execute hardware processing.	Mandatory
uvcs_cmn_set_preempt_mode	Preempt target hardware.	Optional
uvcs_cmn_get_work_size	Gets work size of this library.	Optional
uvcs_cmn_get_ip_info	Gets the information of video hardware.	Mandatory

4.1.2 Callback Function

This library needs callback functions of following.

These callback functions are mainly used for accessing OS dependent function and are set in the function uvcs_cmn_initialize (4.2.2).

The types of each callback and the callback timing are shown in Table 4.2.

Table 4.2 List of System Callback

Types	Description
UVCS_CMN_CB_REG_READ	Reads from hardware register. When read-access is needed in internal process, this callback will be called.
UVCS_CMN_CB_REG_WRITE	Writes to hardware register. When write-access is needed in internal process, this callback will be called.
UVCS_CMN_CB_HW_START	Notifies the timing of hardware starts.
UVCS_CMN_CB_HW_STOP	Notifies the timing of hardware stops.
UVCS_CMN_CB_PROC_DONE	This callback is for notifying the requested hardware processing ends, and request to send result information to requested module.
UVCS_CMN_CB_SEM_LOCK	This callback requests to lock a semaphore created.
UVCS_CMN_CB_SEM_UNLOCK	This callback requests to unlock a semaphore locked.
UVCS_CMN_CB_SEM_CREATE	This callback requests to create a semaphore for mutual exclusion.
UVCS_CMN_CB_SEM_DESTROY	This callback requests to destroy a semaphore created.
UVCS_CMN_CB_THREAD_EVENT	This callback requests to call uvcs_cmn_execute (4.2.7) from a thread created.
UVCS_CMN_CB_THREAD_CREATE	This callback requests to create a worker thread.
UVCS_CMN_CB_THREAD_DESTROY	This callback requests to destroy a thread created.

4.2 External Function Prototypes

4.2.1 uvcs_cmn_interrupt

<Function Prototypes>

```
UVCS_RESULT uvcs_cmn_interrupt(
                UVCS_CMN_LIB_INFO       lib_info ,
                UVCS_U32                base_addr ,
                UVCS_U32                cur_time ,
                ) ;
```

<Input Parameters>

Parameter	Description
lib_info	Set the library information which is returned on initialization.
base_addr	Set the hardware base address interrupted.
cur_time	Current time (for debug). Normally, the user does not have to set this parameter other than debugging.

<Output Parameters>

None

<Function Attribute>

Attributes	Value
Mandatory function	■ Yes / □ No
Categories	□ Synchronous / □ Asynchronous / ■ Other
Call from interrupt	■ Permitted / □ Prohibited
Call from callback	□ Permitted / ■ Prohibited

<Executable State>

State	Permission
Uninitialized-state (3.2.1)	□ Permitted / ■ Prohibited
Initialized-state (3.2.2)	■ Permitted / □ Prohibited
Ready-state (3.2.3)	■ Permitted / □ Prohibited

<Event Notification>

The following callback may be called in this function.

UVCS_CMN_CB_REG_READ (4.3.1)
UVCS_CMN_CB_REG_WRITE (4.3.2)
UVCS_CMN_CB_PROC_DONE (4.3.5)
UVCS_CMN_CB_HW_STOP (4.3.4)
UVCS_CMN_CB_THREAD_EVENT (4.3.10)

<Return Codes>

UVCS_RTN_OK
UVCS_RTN_PARAMETER_ERROR

<Description>
This function is interrupt handler.
Please call this function immediately when the video hardware interrupted.
For the detailed information of hardware interruption, please see the specification of LSI.

Please set the input argument 'base_addr' as the hardware base address interrupted. This hardware base address is same as which was set in 'ip_base_addr' which is the member of the structure of input argument on initialization UVCS_CMN_INIT_PARAM_T (5.2.2).

When the process of this function ends normally, UVCS_RTN_OK is returned.
If it corresponds to at least one of following conditions, UVCS_RTN_PARAMETER_ERROR is returned.

- The argument 'lib_info' is invalid.
- The argument 'base_addr' is not same as 'ip_base_addr'.

<Note>

- This function is not reentrant about same interrupt factor. Don't perform multiplex call of this function about same interrupt factor (base_addr).

<Sample Code>

Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>

UVCS_RESULT (5.1.2)
UVCS_CMN_LIB_INFO (5.2.3)
UVCS_CMN_INIT_PARAM_T (5.2.2)

4.2.2 uvcs_cmn_initialize

<Function Prototypes>

```
UVCS_RESULT uvcs_cmn_initialize(
                UVCS_CMN_INIT_PARAM_T  *init_param ,
                UVCS_CMN_LIB_INFO      *lib_info
                ) ;
```

<Input Parameters>

Parameter	Description
*init_param	Initialization parameter (5.2.2)

<Output Parameters>

Parameter	Description
*lib_info	Stores the pointer of library information for internal use. Please use this pointer for the argument of another function.

<Function Attribute>

Attributes	Value
Mandatory function	■ Yes / □ No
Categories	■ Synchronous / □ Asynchronous / □ Other
Call from interrupt	□ Permitted / ■ Prohibited
Call from callback	□ Permitted / ■ Prohibited

<Executable State>

State	Permission
Uninitialized-state (3.2.1)	■ Permitted / □ Prohibited
Initialized-state (3.2.2)	□ Permitted / ■ Prohibited
Ready-state (3.2.3)	□ Permitted / ■ Prohibited

<Event Notification>

The following callbacks may be called in this function.
UVCS_CMN_CB_REG_READ (4.3.1)
UVCS_CMN_CB_REG_WRITE (4.3.2)
UVCS_CMN_CB_SEM_CREATE (4.3.8)
UVCS_CMN_CB_SEM_DESTROY (4.3.9)
UVCS_CMN_CB_THREAD_CREATE (4.3.11)
UVCS_CMN_CB_THREAD_DESTROY (4.3.12)

<Return Codes>

UVCS_RTN_OK
UVCS_RTN_PARAMETER_ERROR
UVCS_RTN_SYSTEM_ERROR

<Description>
This function is mandatory function and performs to initialize this library and hardware.
For the detailed information for required setting on this function, please see the section 5.2.2.

When the process of this function ends normally, UVCS_RTN_OK is returned.
If it fails in initialization such as RTOS resource allocation, UVCS_RTN_SYSTEM_ERROR is returned.
If it corresponds to at least one of following conditions, UVCS_RTN_PARAMETER_ERROR is returned.

- The argument 'init_param' is NULL.
- The parameter 'init_param->struct_size' is invalid, or some parameter in the structure has invalid value.

<Note>
None.

<Sample Code>
Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>
UVCS_RESULT (5.1.2)
UVCS_CMN_INIT_PARAM_T (5.2.2)
UVCS_CMN_LIB_INFO (5.2.3)

4.2.3 uvcs_cmn_deinitialize

<Function Prototypes>

```
UVCS_RESULT uvcs_cmn_deinitialize(
                UVCS_CMN_LIB_INFO       lib_info ,
                UVCS_BOOL               forced
                ) ;
```

<Input Parameters>

Parameter	Description
lib_info	Library information which is returned on initialization.
forced	In UVCS_TRUE, the control of mutual exclusion is not performed.

<Output Parameters>

None

<Function Attribute>

Attribute	Value
Mandatory function	■ Yes / □ No
Categories	■ Synchronous / □ Asynchronous / □ Other
Call from interrupt	□ Permitted / ■ Prohibited
Call from callback	□ Permitted / ■ Prohibited

<Executable State>

State	Permission
Uninitialized-state (3.2.1)	□ Permitted / ■ Prohibited
Initialized-state (3.2.2)	■ Permitted / □ Prohibited
Ready-state (3.2.3)	□ Permitted / ■ Prohibited

<Event Notification>

The following callbacks may be called in this function.
UVCS_CMN_CB_SEM_LOCK (4.3.6)
UVCS_CMN_CB_SEM_UNLOCK (4.3.7)
UVCS_CMN_CB_SEM_DESTROY (4.3.9)
UVCS_CMN_CB_THREAD_DESTROY (4.3.12)

<Return Codes>

UVCS_RTN_OK
UVCS_RTN_NOT_INITIALIZE
UVCS_RTN_CONTINUE

<Description>
This function performs to de-initialize this library, and changes the state of this library to uninitialized state. It is strongly recommended to destroy all contexts of another library before calling this function.

When the process of this function ends normally, UVCS_RTN_OK is returned.
If this function is called under uninitialized state (the argument 'lib_info' is invalid), UVCS_RTN_NOT_INITIALIZE is returned.

If UVCS_FALSE was returned in the callback of locking mutual exclusion, this library returns UVCS_RTN_CONTINUE. In this case, please retry to call this function. (In the case of 'forced' is UVCS_FALSE)

<Note>
- Please be sure to close all handle. If the user should not do so, it may become a result of memory-leak.

<Sample Code>
Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>
UVCS_RESULT (5.1.2)
UVCS_CMN_LIB_INFO (5.2.3)
UVCS_CMN_CB_SEM_LOCK (4.3.6)
UVCS_CMN_CB_SEM_UNLOCK (4.3.7)
uvcs_cmn_initialize (4.2.2)

4.2.4 uvcs_cmn_open

<Function Prototypes>

```
UVCS_RESULT uvcs_cmn_open(
                UVCS_CMN_LIB_INFO       lib_info ,
                UVCS_CMN_OPEN_PARAM_T  *open_param ,
                UVCS_CMN_HANDLE        *handle
                ) ;
```

<Input Parameters>

Parameter	Description
lib_info	Library information (4.2.2)
*open_param	parameter for opening a handle (5.2.2)

<Output Parameters>

Parameter	Description
*handle	handle (5.2.2)

<Function Attribute>

Attribute	Value
Mandatory function	■ Yes / □ No
Categories	■ Synchronous / □ Asynchronous / □ Other
Call from interrupt	□ Permitted / ■ Prohibited
Call from callback	□ Permitted / ■ Prohibited

<Executable State>

Status	Permission
Uninitialized-state (3.2.1)	□ Permitted / ■ Prohibited
Initialized-state (3.2.2)	■ Permitted / □ Prohibited
Ready-state (3.2.3) *1	■ Permitted / □ Prohibited

*1) not influenced by the state of current handle.

<Event Notification>

The following callbacks may be called in this function.
UVCS_CMN_CB_SEM_LOCK (4.3.6)
UVCS_CMN_CB_SEM_UNLOCK (4.3.7)

<Return Codes>

UVCS_RTN_OK
UVCS_RTN_NOT_INITIALIZE
UVCS_RTN_PARAMETER_ERROR
UVCS_RTN_CONTINUE
UVCS_RTN_BUSY

<Description>
This function allocates handle and prepares processing of hardware. It is necessary to allocate handle for every context of decoder/encoder library. Not to use same handle by two or more decoder/encoder context (not to share).

When the process of this function ends normally, UVCS_RTN_OK is returned.
If this function is called under uninitialized state (the arugment 'lib_info' is invalid), UVCS_RTN_NOT_INITIALIZE is returned.
If UVCS_FALSE was returned in the callback of locking mutual exclusion, this library returnes UVCS_RTN_CONTINUE. In this case, please retry to call this function.
If a user tries use in preempt_mode more than (hw_num-1), UVCS_RTN_BUSY is returned.
If it corresponds to at least one of following conditions, UVCS_RTN_PARAMETER_ERROR is returned.

- The argument 'open_param' is NULL.
- The paremter 'open_param->struct_size' is invalid or some parameter in the structure has invalid value.

<Note>
None.

<Sample Code>
Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>
UVCS_RESULT (5.1.2)
UVCS_CMN_LIB_INFO (5.2.3)
UVCS_CMN_HANDLE (5.2.4)
UVCS_CMN_OPEN_PARAM_T (5.2.5)
UVCS_CMN_CB_SEM_LOCK (4.3.6)
UVCS_CMN_CB_SEM_UNLOCK (4.3.7)
uvcs_cmn_close (4.2.5)

4.2.5 uvcs_cmn_close

<Function Prototypes>

```
UVCS_RESULT uvcs_cmn_close(
                UVCS_CMN_LIB_INFO       lib_info ,
                UVCS_CMN_HANDLE         handle ,
                UVCS_BOOL               forced
                ) ;
```

<Input Parameters>

Parameter	Description
lib_info	Library information
handle	target handle
forced	In UVCS_TRUE, clear an internal state forcibly.

<Output Parameters>

None

<Function Attribute>

Attribute	Value
Mandatory function	■ Yes / □ No
Categories	■ Synchronous / □ Asynchronous / □ Other
Call from interrupt	□ Permitted / ■ Prohibited
Call from callback	□ Permitted / ■ Prohibited

<Executable State>

Status	Permission
Uninitialized-state (3.2.1)	□ Permitted / ■ Prohibited
Initialized-state (3.2.2)	□ Permitted / ■ Prohibited
Ready-state (3.2.3)	■ Permitted / □ Prohibited

<Event Notification>

The following callbacks may be called in this function.

UVCS_CMN_CB_SEM_LOCK (4.3.6)

UVCS_CMN_CB_SEM_UNLOCK (4.3.7)

<Return Codes>

UVCS_RTN_OK

UVCS_RTN_NOT_INITIALIZE

UVCS_RTN_INVALID_HANDLE

UVCS_RTN_CONTINUE

UVCS_RTN_BUSY

<Description>
This function releases allocated handle.

When the process of this function ends normally, UVCS_RTN_OK is returned.
If this function is called under uninitialized state (the argument 'lib_info' is invalid), UVCS_RTN_NOT_INITIALIZE is returned.
If the argument 'handle' is invalid, UVCS_RTN_INVALID_HANDLE is returned.

If UVCS_FALSE was returned in the callback of locking mutual exclusion, this library returnes UVCS_RTN_CONTINUE. In this case, please retry to call this function.
If the target handle is running, UVCS_RTN_BUSY is returned. In this case, please retry to call this function later.

<Note>
None.

<Sample Code>
Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>
UVCS_RESULT (5.1.2)
UVCS_CMN_LIB_INFO (5.2.3)
UVCS_CMN_HANDLE (5.2.4)
UVCS_CMN_CB_SEM_LOCK (4.3.6)
UVCS_CMN_CB_SEM_UNLOCK (4.3.7)

4.2.6 uvcs_cmn_request

<Function Prototypes>

```
UVCS_RESULT uvcs_cmn_request(
                    UVCS_CMN_LIB_INFO                    lib_info ,
                    UVCS_CMN_HANDLE                      handle ,
                    UVCS_CMN_HW_PROC_T                  *req_info
                    ) ;
```

<Input Parameters>

Parameter	Description
lib_info	Library information
handle	Handle for current context.
*req_info	Set the address of the structure in which the request information is stored. Morever, this memory area is used for storing result information.

<Output Parameters>

None

<Function Attribute>

Attribute	Value
Mandatory function	■ Yes / □ No
Categories	■ Synchronous / □ Asynchronous / □ Other
Call from interrupt	□ Permitted / ■ Prohibited
Call from callback	□ Permitted / ■ Prohibited

<Executable State>

Status	Permission
Uninitialized-state (3.2.1)	□ Permitted / ■ Prohibited
Initialized-state (3.2.2)	□ Permitted / ■ Prohibited
Ready-state (3.2.3)	■ Permitted / □ Prohibited

<Event Notification>

The following callbacks may be called in this function.
UVCS_CMN_CB_SEM_LOCK (4.3.6)
UVCS_CMN_CB_SEM_UNLOCK (4.3.7)
UVCS_CMN_CB_THREAD_EVENT (4.3.10)

<Return Codes>

UVCS_RTN_OK
UVCS_RTN_NOT_INITIALIZE
UVCS_RTN_INVALID_HANDLE
UVCS_RTN_PARAMETER_ERROR
UVCS_RTN_CONTINUE
UVCS_RTN_BUSY

<Description>
This function is for requesting hardware processing (decoding/encoding). The reserved processing is started by calling uvcs_cmn_execute (4.2.7). The request to call it is performed via callback UVCS_CMN_CB_THREAD_EVENT (4.3.10).
Since the request information (the argument 'req_info') is used during hardware processing, the user must hold it until the UVCS_CMN_CB_PROC_DONE (4.3.5) event notified.

When the process of this function ends normally, UVCS_RTN_OK is returned.
If this function is called under uninitialized state (argument 'lib_info' is invalid), UVCS_RTN_NOT_INITIALIZE is returned.
If the argument 'handle' is invalid, UVCS_RTN_INVALID_HANDLE is returned.
If UVCS_FALSE was returned in the callback of locking mutual exclusion, this library returns UVCS_RTN_CONTINUE. In this case, please retry to call this function.
When the requesting of processing is not accepted (it does not happen normally), UVCS_RTN_BUSY is returned. In this case, please wait until hardware processing for target handle ends and please retry to call this function after that.
If it corresponds to at least one of following conditions, UVCS_RTN_PARAMETER_ERROR is returned.

- The argument 'req_info' is NULL.
- The parameter 'req_info->struct_size' is invalid or some parameter in the structure has invalid value.

<Note>

- The request information (the argument 'req_info') must be hold until the UVCS_CMN_CB_PROC_DONE (4.3.5) event notification.

<Sample Code>

Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>

UVCS_RESULT (5.1.2)
UVCS_CMN_LIB_INFO (5.2.3)
UVCS_CMN_HANDLE (5.2.4)
UVCS_CMN_HW_PROC_T (5.2.6)
UVCS_CMN_CB_SEM_LOCK (4.3.6)
UVCS_CMN_CB_SEM_UNLOCK (4.3.7)
UVCS_CMN_CB_THREAD_EVENT (4.3.10)

4.2.7 uvcs_cmn_execute

<Function Prototypes>

```
UVCS_RESULT uvcs_cmn_execute(
                UVCS_CMN_LIB_INFO                   lib_info ,
                UVCS_U32                            cur_time
                ) ;
```

<Input Parameters>

Parameter	Description
lib_info	Library information
cur_time	Current time (for debug). Normally, the user does not have to set this parameter other than debugging.

<Output Parameters>

None

<Function Attribute>

Attribute	Value
Mandatory function	■ Yes / □ No
Categories	■ Synchronous / □ Asynchronous / □ Other
Call from interrupt	□ Permitted / ■ Prohibited
Call from callback	□ Permitted / ■ Prohibited

<Executable State>

Status	Permission
Uninitialized-state (3.2.1)	□ Permitted / ■ Prohibited
Initialized-state (3.2.2)	■ Permitted / □ Prohibited
Ready-state (3.2.3)	■ Permitted / □ Prohibited

<Event Notification>

The following callbacks may be called in this function.

UVCS_CMN_CB_REG_READ (4.3.1)
UVCS_CMN_CB_REG_WRITE (4.3.2)
UVCS_CMN_CB_HW_START (4.3.3)
UVCS_CMN_CB_SEM_LOCK (4.3.6)
UVCS_CMN_CB_SEM_UNLOCK (4.3.7)

<Return Codes>

UVCS_RTN_OK
UVCS_RTN_NOT_INITIALIZE
UVCS_RTN_CONTINUE

<Description>
This function searches a request of hardware processing and starts hardware processing (decoding / encoding).
Generally please call this function from a thread created. The request to call this function is performed via callback UVCS_CMN_CB_THREAD_EVENT (4.3.10). When two or more requests received from driver before calling this function, please call once.
This function is for searching a request of hardware processing and is for start hardware processing (decoding / encoding).

When the process of this function ends normally, UVCS_RTN_OK is returned.
If this function is called under uninitialized state (argument 'lib_info' is invalid), UVCS_RTN_NOT_INITIALIZE is returned.
If UVCS_FALSE was returned in the callback of locking mutual exclusion, this function returns UVCS_RTN_CONTINUE. In this case, please retry to call this function.

<Note>
None.

<Sample Code>
Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>
UVCS_RESULT (5.1.2)
UVCS_CMN_LIB_INFO (5.2.3)
UVCS_CMN_CB_REG_READ (4.3.1)
UVCS_CMN_CB_REG_WRITE (4.3.2)
UVCS_CMN_CB_HW_START (4.3.3)
UVCS_CMN_CB_SEM_LOCK (4.3.6)
UVCS_CMN_CB_SEM_UNLOCK (4.3.7)

4.2.8 uvcs_cmn_set_preempt_mode

<Function Prototypes>

```
UVCS_RESULT uvcs_cmn_set_preempt_mode(
                    UVCS_CMN_LIB_INFO       lib_info ,
                    UVCS_CMN_HANDLE         handle ,
                    UVCS_BOOL               preempt_mode ,
                    UVCS_U32                preempt_hwid
                    ) ;
```

<Input Parameters>

Parameter	Description
lib_info	Set the library information which is returned on initialization.
handle	Target handle.
preempt_mode	UVCS_TRUE: Enable preempt mode, UVCS_FALSE: Disable it.
preempt_hwid	Set the target hardware number. (default 0)

<Output Parameters>

None

<Function Attribute>

Attributes	Value
Mandatory function	□ Yes / ■ No
Categories	■ Synchronous / □ Asynchronous / □ Other
Call from interrupt	□ Permitted / ■ Prohibited
Call from callback	□ Permitted / ■ Prohibited

<Executable State>

State	Permission
Uninitialized-state (3.2.1)	□ Permitted / ■ Prohibited
Initialized-state (3.2.2)	□ Permitted / ■ Prohibited
Ready-state (3.2.3)	■ Permitted / □ Prohibited

<Event Notification>

The following callback may be called in this function.

UVCS_CMN_CB_SEM_LOCK (4.3.6)

UVCS_CMN_CB_SEM_UNLOCK (4.3.7)

<Return Codes>

UVCS_RTN_OK

UVCS_RTN_INVALID_HANDLE

UVCS_RTN_CONTINUE

UVCS_RTN_BUSY

UVCS_RTN_PARAMETER_ERROR

<Description>
This function is for changing the preempt mode of target handle. If the preempt mode is UVCS_TRUE, the target hardware is preempted by this handle. The functional of this mode is the same as the preempt_mode which is set in the uvcs_cmn_open (4.2.4).

When the process of this function ends normally, UVCS_RTN_OK is returned.
If the argument 'handle' is invalid, UVCS_RTN_INVALID_HANDLE is returned.
If the number of video-hardware controlled by this library is 1, or if another handle has used this mode, UVCS_RTN_BUSY is returned.
If UVCS_FALSE was returned in the callback of locking mutual exclusion, UVCS_RTN_CONTINUE is returned.
If it corresponds to at least one of following conditions, UVCS_RTN_PARAMETER_ERROR is returned.

- the argument 'preempt_hwid' has invalid value.

<Note>
None.

<Sample Code>
Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>
UVCS_RESULT (5.1.2)
UVCS_CMN_LIB_INFO (5.2.3)
UVCS_CMN_HANDLE (5.2.4)

4.2.9 uvcs_cmn_get_work_size

<Function Prototypes>

```
UVCS_RESULT uvcs_cmn_get_work_size (
                UVCS_U32              *work_mem_0_size ,
                UVCS_U32              *hdl_work_0_size
                ) ;
```

<Input Parameters>

None

<Output Parameters>

Parameter	Description
*work_mem_0_size	Stores the work size of this library.
*hdl_work_0_size	Stores the work size of every handle.

<Function Attribute>

Attributes	Value
Mandatory function	□ Yes / ■ No
Categories	■ Synchronous / □ Asynchronous / □ Other
Call from interrupt	□ Permitted / ■ Prohibited
Call from callback	□ Permitted / ■ Prohibited

<Executable State>

State	Permission
Uninitialized-state (3.2.1)	■ Permitted / □ Prohibited
Initialized-state (3.2.2)	□ Permitted / ■ Prohibited
Ready-state (3.2.3)	□ Permitted / ■ Prohibited

<Event Notification>

None

<Return Codes>

UVCS_RTN_OK
UVCS_RTN_PARAMETER_ERROR

<Description>
This function is for getting the work size of this library. .

When the process of this function ends normally, UVCS_RTN_OK is returned.
If it corresponds to at least one of following conditions, UVCS_RTN_PARAMETER_ERROR is returned.

- The argument ‘work_mem_0_size’ or ‘hdl_work_0_size’ is NULL.

<Note>
None.

<Sample Code>
Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>
UVCS_RESULT (5.1.2)

4.2.10 uvcs_cmn_get_ip_info

<Function Prototypes>

```
UVCS_RESULT uvcs_cmn_get_ip_info(
                    UVCS_CMN_LIB_INFO                    lib_info ,
                    UVCS_CMN_IP_INFO_T                  *ip_info
                    ) ;
```

<Input Parameters>

Parameter	Description
lib_info	Library information
*ip_info	Set the address of the structure UVCS_CMN_IP_INFO_T, and set the size to ip_info->struct_size.

<Output Parameters>

Parameter	Description
*ip_info	Stores the hardware information.

<Function Attribute>

Attribute	Value
Mandatory function	■ Yes / □ No
Categories	■ Synchronous / □ Asynchronous / □ Other
Call from interrupt	□ Permitted / ■ Prohibited
Call from callback	□ Permitted / ■ Prohibited

<Executable State>

Status	Permission
Uninitialized-state (3.2.1)	□ Permitted / ■ Prohibited
Initialized-state (3.2.2)	■ Permitted / □ Prohibited
Ready-state (3.2.3)	■ Permitted / □ Prohibited

<Event Notification>

None

<Return Codes>

UVCS_RTN_OK
UVCS_RTN_NOT_INITIALIZE
UVCS_RTN_PARAMETER_ERROR

<Description>
This function is for reading the information of video hardware. This information is used for determining hardware functional difference by the decoder library and encoder library.

When the process of this function ends normally, UVCS_RTN_OK is returned.
If this function is called under uninitialized state (argument 'lib_info' is invalid), UVCS_RTN_NOT_INITIALIZE is returned.
If it corresponds to at least one of following conditions, UVCS_RTN_PARAMETER_ERROR is returned.

- The argument 'ip_info' is NULL.
- The argument 'ip_info->struct_size' has invalid value.

<Note>
None.

<Sample Code>
Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>
UVCS_RESULT (5.1.2)

4.3 System Callback

4.3.1 UVCS_CMN_CB_REG_READ

<Function Prototypes>

```
typedef void (*UVCS_CMN_CB_REG_READ)(
                UVCS_PTR                udptr ,
                volatile UVCS_U32      *reg_addr ,
                UVCS_U32               *dst_addr ,
                UVCS_U32                num_reg
                ) ;
```

<Input Parameters>

Parameter	Description
udptr	Stores the user-defined pointer which is set on initialization.
*reg_addr	Stores the target address of register to read.
*dst_addr	Stores the destination address for storing read-out data.
num_reg	Stores number of registers to read.

<Output Parameters>

Parameter	Description
*dst_addr	Please set read-out data.

<Return Value>

None

<Description>

This callback is for reading hardware register by video driver. The registration of this callback is done in initialization function uvcs_cmn_initialize (4.2.2). Please set the function pointer to 'cb_reg_read' which is the member of structure of input argument UVCS_CMN_INIT_PARAM_T (5.2.2). The registration of this callback is mandatory.

- Expected Operations

Read the 'num_reg' registers specified by '*reg_addr', and store it to '*dst_addr'.

<Note>

- Since it becomes a cause of low throughput, please do these processes as soon as possible in this callback. And since this callback may be called from interrupt, please design it carefully.

<Sample Code>

Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>

UVCS_CMN_INIT_PARAM_T (5.2.2)

4.3.2 UVCS_CMN_CB_REG_WRITE

<Function Prototypes>

```
typedef void (*UVCS_CMN_CB_REG_WRITE)(
                UVCS_PTR                udptr ,
                volatile UVCS_U32      *reg_addr ,
                UVCS_U32               *src_addr ,
                UVCS_U32                num_reg
                ) ;
```

<Input Parameters>

Parameter	Description
udptr	Stores the user-defined pointer which is set on initialization.
*reg_addr	Stores the target address of register to write.
*src_addr	Stores the source address of data to write.
num_reg	Stores number of registers to write.

<Output Parameters>

None

<Return Value>

None

<Description>

This callback is for writing data to hardware register by video driver. The registration of this callback is done in initialization function uvcs_cmn_initialize (4.2.2). Please set the function pointer to 'cb_reg_write' which is the member of structures of input argument UVCS_CMN_INIT_PARAM_T (5.2.2). The registration of this callback is mandatory.

- Expected Operations

Write 'num_reg' data specified by '*src_addr', to hardware register specified by '*reg_addr'.

<Note>

- Since it becomes a cause of low throughput, please do these processes as soon as possible in this callback. And since this callback may be called from interrupt, please design it carefully.

<Sample Code>

Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>

UVCS_CMN_INIT_PARAM_T (5.2.2)
UVCS_CMN_CB_REG_READ (4.3.1)

4.3.3 UVCS_CMN_CB_HW_START

<Function Prototypes>

```
typedef void (*UVCS_CMN_CB_HW_START)(
                UVCS_PTR              udptr ,
                UVCS_U32              hw_ip_id ,
                UVCS_U32              hw_module_id ,
                UVCS_U32             *baa
                ) ;
```

<Input Parameters>

Parameter	Description
udptr	Stores the user-defined pointer which is set on initialization.
hw_ip_id	Stores the hardware identifier to start processing.
hw_module_id	Stores the hardware module identifier to start processing.
baa	Stores the base address information (Internally used).

<Output Parameters>

None

<Return Value>

None

<Description>

Notifies the timing of hardware starts. The registration of this callback is done in initialization function uvcs_cmn_initialize (4.2.2). Please set the function pointer to 'cb_hw_start' which is the member of structure of input argument UVCS_CMN_INIT_PARAM_T (5.2.2). If the user wants to control power consumption, it is possible to control it by using the arguments of this callback.

The registration of this callback is not indispensable.

<Note>

- Since there is a possibility that it is called from interrupt context, please design it carefully.

<Sample Code>

Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>

UVCS_CMN_INIT_PARAM_T(5.2.2)

4.3.4 UVCS_CMN_CB_HW_STOP

<Function Prototypes>

```
typedef void (*UVCS_CMN_CB_HW_STOP)(
                UVCS_PTR                udptr ,
                UVCS_U32                hw_ip_id ,
                UVCS_U32                hw_module_id
                ) ;
```

<Input Parameters>

Parameter	Description
udptr	Stores the user-defined pointer which is set on initialization.
hw_ip_id	Stores the hardware identifier to start processing.
hw_module_id	Stores the hardware module identifier to start processing.

<Output Parameters>

None

<Return Value>

None

<Description>

Notifies the timing of hardware stops. The registration of this callback is done in initialization function uvcs_cmn_initialize (4.2.2). Please set the function pointer to 'cb_hw_stop' which is the member of structure of input argument UVCS_CMN_INIT_PARAM_T (5.2.2). If the user wants to control power consumption, it is possible to control it by using the arguments of this callback.

The registration of this callback is not indispensable.

<Note>

- Since there is a possibility that it is called from interrupt context, please design it carefully.

<Sample Code>

Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>

UVCS_CMN_INIT_PARAM_T(5.2.2)

4.3.5 UVCS_CMN_CB_PROC_DONE

<Function Prototypes>

```
typedef void (*UVCS_CMN_CB_PROC_DONE)(
                UVCS_PTR                    udptr ,
                UVCS_PTR                    hdl_udptr ,
                UVCS_CMN_HANDLE             handle ,
                UVCS_CMN_HW_PROC_T          *res_info
                ) ;
```

<Input Parameters>

Parameter	Description
udptr	Stores the user-defined pointer which is set on uvcs_cmn_initialize (4.2.2).
hdl_udptr	Stores the user-defined pointer which is set on uvcs_cmn_open (4.2.4).
handle	Stores the target handle.
*req_info	Stores the result information.

<Output Parameters>

None

<Return Value>

None

<Description>

This callback is for notifying the requested hardware processing ends, and request to send result information to requested module.
Please send result information which is stored in the argument 'res_info->cmd_param'. Since this result information is discarded after this callback, please copy it if needed. Because this event may be notified continously (the maximum number of continuous callback is same as UVCS_CMN_PROC_REQ_MAX).

<Note>

- Since there is a possibility that it is called from interrupt context, please design it carefully.

<Sample Code>

Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>

UVCS_CMN_INIT_PARAM_T (5.2.2)
UVCS_CMN_HANDLE (5.2.4)
UVCS_CMN_HW_PROC_T (5.2.6)

4.3.6 UVCS_CMN_CB_SEM_LOCK

<Function Prototypes>

```
typedef UVCS_BOOL (*UVCS_CMN_CB_SEM_LOCK)( UVCS_PTR udptr ) ;
```

<Input Parameters>

Parameter	Description
udptr	Stores the user-defined pointer which is set on uvcs_cmn_initialize (4.2.2).

<Output Parameters>

None

<Return Value>

Value	Description
UVCS_TRUE	Successed
UVCS_FALSE	Failed to lock a semaphore.

<Description>

This callback requests to lock a semaphore for mutual exclusion.
If UVCS_FALSE is returned from user in this callback, the current function will return UVCS_RTN_CONTINUE. In this case, please retry to call current function.

<Note>

None.

<Sample Code>

Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>

UVCS_CMN_INIT_PARAM_T (5.2.2)

4.3.7 UVCS_CMN_CB_SEM_UNLOCK

<Function Prototypes>

```
typedef void (*UVCS_CMN_CB_SEM_UNLOCK)( UVCS_PTR udptr ) ;
```

<Input Parameters>

Parameter	Description
udptr	Stores the user-defined pointer which is set on uvcs_cmn_initialize (4.2.2).

<Output Parameters>

None

<Return Value>

None

<Description>

This callback requests to unlock a semaphore locked.

<Note>

None.

<Sample Code>

Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>

UVCS_CMN_INIT_PARAM_T (5.2.2)

4.3.8 UVCS_CMN_CB_SEM_CREATE

<Function Prototypes>

```
typedef UVCS_BOOL (*UVCS_CMN_CB_SEM_CREATE)( UVCS_PTR udptr ) ;
```

<Input Parameters>

Parameter	Description
udptr	Stores the user-defined pointer which is set on uvcs_cmn_initialize (4.2.2).

<Output Parameters>

None

<Return Value>

Value	Description
UVCS_TRUE	Successed
UVCS_FALSE	Failed to lock a semaphore.

<Description>

This callback requests to create a semaphore. The required number of semaphore is 1. It is not needed to lock it on the timing of creating.

<Note>

None.

<Sample Code>

Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>

UVCS_CMN_INIT_PARAM_T (5.2.2)

4.3.9 UVCS_CMN_CB_SEM_DESTROY

<Function Prototypes>

```
typedef void (*UVCS_CMN_CB_SEM_DESTROY)( UVCS_PTR udptr ) ;
```

<Input Parameters>

Parameter	Description
udptr	Stores the user-defined pointer which is set on uvcs_cmn_initialize (4.2.2).

<Output Parameters>

None

<Return Value>

None

<Description>

This callback requests to destroy a semaphore created.

<Note>

None.

<Sample Code>

Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>

UVCS_CMN_INIT_PARAM_T (5.2.2)

4.3.10 UVCS_CMN_CB_THREAD_EVENT

<Function Prototypes>

```
typedef void (*UVCS_CMN_CB_THREAD_EVENT)( UVCS_PTR udptr ) ;
```

<Input Parameters>

Parameter	Description
udptr	Stores the user-defined pointer which is set on uvcs_cmn_initialize (4.2.2).

<Output Parameters>

None

<Return Value>

None

<Description>

This callback requests to call the function uvcs_cmn_execute (4.2.7) from thread created.

<Note>

- Since there is a possibility that it is called from interrupt context, please design it carefully.

<Sample Code>

Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>

UVCS_CMN_INIT_PARAM_T (5.2.2)

4.3.11 UVCS_CMN_CB_THREAD_CREATE

<Function Prototypes>

```
typedef UVCS_BOOL (*UVCS_CMN_CB_THREAD_CREATE)( UVCS_PTR udptr ) ;
```

<Input Parameters>

Parameter	Description
udptr	Stores the user-defined pointer which is set on uvcs_cmn_initialize (4.2.2).

<Output Parameters>

None

<Return Value>

Value	Description
UVCS_TRUE	Successed
UVCS_FALSE	Failed to create thread.

<Description>

This callback requests to create worker thread. This callback is called once in uvcs_cmn_initialize (4.2.2).
If UVCS_FALSE is returned from this callback, the initialization function will return UVCS_RTN_SYSTEM_ERROR.

<Note>

None.

<Sample Code>

Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>

UVCS_CMN_INIT_PARAM_T (5.2.2)

4.3.12 UVCS_CMN_CB_THREAD_DESTROY

<Function Prototypes>

```
typedef void (*UVCS_CMN_CB_THREAD_DESTROY)( UVCS_PTR udptr ) ;
```

<Input Parameters>

Parameter	Description
udptr	Stores the user-defined pointer which is set on uvcs_cmn_initialize (4.2.2).

<Output Parameters>

None

<Return Value>

None

<Description>

This callback requests to destroy worker thread. This callback is called once in uvcs_cmn_deinitialize (4.2.3).

<Note>

None.

<Sample Code>

Please refer the Sample Codes Manual of UVCS-CMN.

<See Also>

UVCS_CMN_INIT_PARAM_T (5.2.2)

5. Types

5.1 UVCS Common

This library uses following types defined in common header file “uvcs_types.h”.

5.1.1 Basic type

This section shows the basic type used on UVCS.

Table 5.1 Basic Type

Types	Definition	Basic types
UVCS_U8	typedef	unsigned char
UVCS_S32	typedef	signed long
UVCS_U32	typedef	unsigned long
UVCS_FALSE	Enumerator (UVCS_BOOL)	0
UVCS_TRUE	Enumerator (UVCS_BOOL)	1
UVCS_PTR	typedef	void *

5.1.2 UVCS_RESULT

This enumerator is common return code on UVCS and defined in “uvcs_types.h”.
In this library this enumerator is used for return value in the external function.

```
typedef enum {
        UVCS_RTN_OK                     = 0x00L,
        UVCS_RTN_INVALID_HANDLE         = 0x01L,
        UVCS_RTN_INVALID_STATE          = 0x02L,
        UVCS_RTN_PARAMETER_ERROR        = 0x03L,
        UVCS_RTN_NOT_SUPPORTED          = 0x04L,
        UVCS_RTN_NOT_CONFIGURED         = 0x05L,
        UVCS_RTN_NOT_INITIALIZE         = 0x06L,
        UVCS_RTN_ALREADY_INITIALIZED    = 0x07L,
        UVCS_RTN_SYSTEM_ERROR           = 0x08L
        UVCS_RTN_BUSY                   = 0x09L,
        UVCS_RTN_CONTINUE               = 0x0AL,
} UVCS_RESULT ;
```

Table 5.2 Enumerator UVCS_RESULT

Value	Description
UVCS_RTN_OK	Success
UVCS_RTN_INVALID_HANDLE	When the target handle is not valid, this value will be returned.
UVCS_RTN_INVALID_STATE	Unused
UVCS_RTN_PARAMETER_ERROR	Insufficient parameter setting or invalid parameter setting.
UVCS_RTN_NOT_SUPPORTED	Unused
UVCS_RTN_NOT_CONFIGURED	Unused
UVCS_RTN_NOT_INITIALIZE	The function which cannot be called on the Uninitialized-state (3.2.1) was called.
UVCS_RTN_ALREADY_INITIALIZED	Unused
UVCS_RTN_SYSTEM_ERROR	Fatal error. This value will be returned in following case; ● When the initialization process for hardware is failed. ● When the semaphore creating is failed. ● When the thread creating is failed.
UVCS_RTN_BUSY	This value will be returned in following case; ● When the request of hardware processing could not be accepted. In this case, please retry to call current function. ● If it tries to preempt target hardware when already preempted hardware exist.
UVCS_RTN_CONTINUE	This value will be returned when the callback function for disabling interruption is returned UVCS_FALSE, for the reason of interrupt etc. In this case, please retry to call current function.

5.2 Type definitions for this library

5.2.1 Definitions

This section shows the definition value used on this library.

Table 5.3 Definitions

Name	Value	Description
UVCS_CMN_MAX_HW_NUM	2uL	Indicates the maximum number of hardware which can be handled by this library.
UVCS_CMN_HW_MODULE_NUM	2uL	Indicates the number of hw modules per video hardware.
UVCS_CMN_PROC_REQ_MAX	3uL	The maximum number of hardware processing request for each handle. The maximum number of continuous callback notification of the end of processing is same as this value. If your system could not send callback notification to source module, it is required to save the callback data.
UVCS_CMN_HWP_CMD_NOEL	128uL	Indicates the number of element of array in the structure UVCS_CMN_HW_PROC_T.

5.2.2 UVCS_CMN_INIT_PARAM_T

This structure is the configurations for initialization and is used as a parameter on uvcs_cmn_initialize (4.2.2).

```
typedef struct {
        UVCS_U32                struct_size ;
        UVCS_U32                hw_num ;
        UVCS_PTR                udptr ;
        /* work memory */
        UVCS_U32 *              work_mem_0_virt ;
        UVCS_U32                work_mem_0_size ;
        /* ip information */
        UVCS_U32                ip_base_addr[ UVCS_CMN_MAX_HW_NUM ][ UVCS_CMN_HW_MODULE_NUM ] ;
        UVCS_U32                ip_option ;
        /* system callback */
        UVCS_CMN_CB_REG_READ            cb_reg_read ;
        UVCS_CMN_CB_REG_WRITE           cb_reg_write ;
        UVCS_CMN_CB_HW_START            cb_hw_start ;
        UVCS_CMN_CB_HW_STOP             cb_hw_stop ;
        UVCS_CMN_CB_PROC_DONE           cb_proc_done ;
        UVCS_CMN_CB_SEM_LOCK            cb_sem_lock ;
        UVCS_CMN_CB_SEM_UNLOCK          cb_sem_unlock ;
        UVCS_CMN_CB_SEM_CREATE          cb_sem_create ;
        UVCS_CMN_CB_SEM_DESTROY         cb_sem_destroy ;
        UVCS_CMN_CB_THREAD_EVENT        cb_thr_event ;
        UVCS_CMN_CB_THREAD_CREATE       cb_thr_create ;
        UVCS_CMN_CB_THREAD_DESTROY      cb_thr_destroy ;
        /* for debug */
        UVCS_U32 *                      debug_log_buff ;
        UVCS_U32                        debug_log_size ;
} UVCS_CMN_INIT_PARAM_T ;
```

Table 5.4 Structure of UVCS_CMN_INIT_PARAM_T

Member	R/W	Description
struct_size	W	Set the size of this structure in byte unit.
hw_num	W	Set the maximum number of hardware handling by this library. Please refer LSI specifications before setting the value for this parameter. The effective value in this parameter is smaller than UVCS_CMN_MAX_HW_NUM.
udptr	W	Set the user-define pointer. This pointer will be returned with callback.
work_mem_0_virt	W	Set the virtual address value of the top address of memory area for this library. For the detailed information for this memory area (such as alignment), please refer 2.3.
work_mem_0_size	W	Set the memory size of work_mem_0. For the detailed information for the memory size, please refer 2.3.

<table>
<tr><th>Member</th><th>R/W</th><th>Description</th></tr>
<tr><td>ip_base_addr[n] [m]</td><td>W</td><td>Set the base address of video hardware. Please refer the address-map of LSI for setting this parameter.
<table>
<tr><th colspan="2">Index</th><th>Description</th></tr>
<tr><td colspan="2">n</td><td>Video hardware number</td></tr>
<tr><td>m</td><td>0</td><td>Base address for video hardware #n module 0. If the target LSI has two or more video hardwares, please set up a parameter of the rest of this array. Otherwise please set 0 to the rest of this array.</td></tr>
<tr><td></td><td>1</td><td>Base address for video hardware #n module 1.</td></tr>
</table></td></tr>
<tr><td>ip_option</td><td>W</td><td>Set the optional settings of video hardware. (Default = 0x0002000AuL)
Please set the default value, as long as there is no special information from us.</td></tr>
<tr><td>cb_reg_read</td><td>W</td><td>Set the function pointer for callback to read from hardware register. The specification of callback function is shown in section 4.3.1.</td></tr>
<tr><td>cb_reg_write</td><td>W</td><td>Set the function pointer for callback to write to hardware register. The specification of callback function is shown in section 4.3.2.</td></tr>
<tr><td>cb_hw_start</td><td>W</td><td>Set the function pointer for callback to notify the timing of hardware starts. If the user want to control power consumption, it is possible to control it by using its function arguments. The specification of callback function is shown in the section 4.3.3. If the user doesn’t need this callback, please set NULL.</td></tr>
<tr><td>cb_hw_stop</td><td>W</td><td>Set the function pointer for callback to notify the timing of hardware stops. If the user wants to control power consumption, it is possible to control it by using its function arguments. The specification of callback function is shown in the section 4.3.4. If the user doesn’t need this callback, please set NULL.</td></tr>
<tr><td>cb_proc_done</td><td>W</td><td>Set the function pointer for callback to notify the ends of hardware processing. The specification of this callback is shown in section 4.3.5.</td></tr>
<tr><td>cb_sem_lock</td><td>W</td><td>Set the function pointer for callback to lock created semaphore. The specification of this callback is shown in section 4.3.6.</td></tr>
<tr><td>cb_sem_unlock</td><td>W</td><td>Set the function pointer for callback to unlock semaphore. The specification of this callback is shown in the section 4.3.6.</td></tr>
<tr><td>cb_sem_create</td><td>W</td><td>Set the function pointer for callback to create a semaphore. The specification of this callback is shown in the section 4.3.6.</td></tr>
<tr><td>cb_sem_destroy</td><td>W</td><td>Set the function pointer for callback to destroy created semaphore. The specification of this callback is shown in the section 4.3.6.</td></tr>
<tr><td>cb_thr_event</td><td>W</td><td>Set the function pointer for callback to request to call uvcs_cmn_execute (4.2.7) from the thread created. The specification of this callback is shown in the section 4.3.10.</td></tr>
<tr><td>cb_thr_create</td><td>W</td><td>Set the function pointer for callback to create a worker thread. The specification of this callback is shown in the section 4.3.11.</td></tr>
<tr><td>cb_thr_destroy</td><td>W</td><td>Set the function pointer for callback to destroy a worker thread created. The specification of this callback is shown in the section 4.3.12.</td></tr>
<tr><td>debug_log_buff</td><td>W</td><td>For debugging use. When the debugging information is needed, please set the memory area for log output. Otherwise, please set NULL.</td></tr>
<tr><td>debug_log_size</td><td>W</td><td>Set the size of the memory area for log output.</td></tr>
</table>

5.2.3 UVCS_CMN_LIB_INFO

This type-definition is used for the global information for this library and used as an argument of all functions in this library.

```
typedef UVCS_U32 * UVCS_CMN_LIB_INFO ;
```

5.2.4 UVCS_CMN_HANDLE

This type-definition is used for the context-handle and used as an argument of the function 4.2.4 , 4.2.5, 4.2.6, and 4.2.8.

```
typedef UVCS_U32 * UVCS_CMN_HANDLE ;
```

5.2.5 UVCS_CMN_OPEN_PARAM_T

This structure is the parameter for opening a handle and used as an argument on uvcs_cmn_open (4.2.4).

```
typedef struct {
        UVCS_U32        struct_size ;
        UVCS_PTR        hdl_udptr ;
        UVCS_U32 *      hdl_work_0_virt ;
        UVCS_U32        hdl_work_0_size ;
        UVCS_BOOL       preempt_mode ;
        UVCS_U32        preempt_hwid ;
} UVCS_CMN_OPEN_PARAM_T ;
```

Table 5.5 Structure of UVCS_CMN_OPEN_PARAM_T

Member	R/W	Description
struct_size	W	Set the size of this structure in byte unit.
hdl_udptr	W	Set the user-defined-pointer value. This value will be returned on the event callback.
hdl_work_0_virt	W	Set the virtual address value of the top address of memory area for new handle. For the detailed information for this memory area (such as alignment), please see 2.3.
hdl_work_0_size	W	Set the memory size of hdl_work_0. For the detailed information for memory size, please see 2.3.
preempt_mode	W	When this parameter is UVCS_TRUE, the target hardware is preempted by this handle. If the number of video-hardware controlled by this library is 1, or if the other handle has used this mode, this mode is unable to use. In this case, the function will return UVCS_RTN_BUSY.
preempt_hwid	W	Set the target hardware number. This parameter is effective only when the preempt_mode is UVCS_TRUE.

5.2.6 UVCS_CMN_HW_PROC_T

This structure is the request parameter for hardware processing and used as an argument on uvcs_cmn_request (4.2.6). And, this structure is the result parameters of hardware processing and used as an argument on the callback UVCS_CMN_CB_PROC_DONE (4.3.5). Unless there are special reasons, all parameters in this structure shall be copied from request information structure from decoder or encoder library or shall be copied to result information structure of decoder or encoder library. The detailed information of these structures, please see the User's Manual of UVCS-DEC / UVCS-ENC.

```
#define     UVCS_CMN_HWP_CMD_NOEL           (128uL)
typedef struct {
        UVCS_U32                    struct_size ;
        UVCS_U32                    module_id ;
        UVCS_U32                    req_serial ;
        UVCS_U32                    cmd_param_noel ;
        UVCS_U32                    cmd_param[ UVCS_CMN_HWP_CMD_NOEL ] ;
} UVCS_CMN_HW_PROC_T ;
```

Table 5.6 Structure of UVCS_CMN_HW_PROC_T

Member	R/W	Description
struct_size	R/W	Set / Stores the size of this structure in byte unit.
module_id	R/W	Set the target hardware module identifier which was received from UVCS-DEC. Stores the hardware module identifier of which hardware processing was finished.
req_serial	R/W	Set the arbitrary identification number of request hardware processing. Stores the identification number which was set at uvcs_cmn_request (4.2.6).
cmd_param_noel	R/W	Set / Stores the effective number of element of cmd_param.
cmd_param	R/W	Set the execution parameters for hardware processing that were requested from UVCS-DEC / UVCS-ENC. Or stores the result parameters of hardware processing that should be sent to UVCS-DEC/ UVCS-ENC library. For more detail, please refer the User's Manual of UVCS-DEC / UVCS-ENC.

5.2.7 UVCS_CMN_IP_INFO_T

This structure indicates the hardware information and is used as an argument on uvcs_cmn_get_ip_info (4.2.10).

In the UVCS-DEC / UVCS-ENC library, this information can be used without modifying, because the structure UVCS_DEC_IP_INFO_T / UVCS_ENC_IP_INFO_T are same as this structure.

```
typedef struct {
        UVCS_U32                    struct_size ;
        UVCS_U32                    ip_version ;
        UVCS_U32                    ip_option ;
} UVCS_CMN_IP_INFO_T ;
```

Table 5.7 Structure of UVCS_CMN_IP_INFO_T

Member	R/W	Description
struct_size	R/W	Set / Stores the size of this structure in byte unit.
ip_version	R	Stores the version information of video hardware.
ip_option	R	Stores the optional settings of video hardware. This value is same as ip_option variable which was set on uvcs_cmn_initialize (4.2.2).

REVISION HISTORY	UVCS Common Engine Library Functions Manual

Rev.	date	revision	
		pages	point
1.00	2014.08.05	ー	version 1.0

UVCS Common Engine Library
Functions Manual

Publication Date Rev.1.00 Aug 05, 2014

Published by Renesas Electronics Corporation

UVCS Common Engine Library

Functions Manual

Renesas Electronics Corporation