International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

F.735.2

(06/2021)

SERIES F: NON-TELEPHONE TELECOMMUNICATION SERVICES

Multimedia services

Architecture and protocols for software-defined cameras

Recommendation ITU-T F.735.2

ITU-T F-SERIES RECOMMENDATIONS

NON-TELEPHONE TELECOMMUNICATION SERVICES

TELEGRAPH SERVICE	
Operating methods for the international public telegram service	F.1–F.19
The gentex network	F.20–F.29
Message switching	F.30–F.39
The international telemessage service	F.40–F.58
The international telex service	F.59–F.89
Statistics and publications on international telegraph services	F.90–F.99
Scheduled and leased communication services	F.100–F.104
Phototelegraph service	F.105–F.109
MOBILE SERVICE	
Mobile services and multidestination satellite services	F.110–F.159
TELEMATIC SERVICES	
Public facsimile service	F.160–F.199
Teletex service	F.200–F.299
Videotex service	F.300–F.349
General provisions for telematic services	F.350–F.399
MESSAGE HANDLING SERVICES	F.400–F.499
DIRECTORY SERVICES	F.500–F.549
DOCUMENT COMMUNICATION	
Document communication	F.550–F.579
Programming communication interfaces	F.580–F.599
DATA TRANSMISSION SERVICES	F.600–F.699
MULTIMEDIA SERVICES	**F.700–F.799**
ISDN SERVICES	F.800–F.849
UNIVERSAL PERSONAL TELECOMMUNICATION	F.850–F.899
ACCESSIBILITY AND HUMAN FACTORS	F.900–F.999

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T F.735.2

Architecture and protocols for software-defined cameras

Summary

As defined in Recommendation ITU-T F.735.1, software-defined cameras can provide the basic hardware service application programming interfaces (APIs) and the common software service APIs for application developers; these APIs are called "service-oriented interfaces". Through these APIs the upper layer applications can obtain raw video, pan/tilt/zoom (PTZ), common software and hardware resource to realize video intelligent analysis and non-intelligent businesses. Recommendation ITU-T F.735.2 specifies the architecture and service-oriented interface protocols for software-defined cameras, including functional architecture of software-defined camera system, service-oriented interface message protocol structure, and service-oriented interface protocols.

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T F.735.2	2021-06-13	16	11.1002/1000/14678

Keywords

Service-oriented interface, software-defined camera.

* To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents/software copyrights, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the appropriate ITU-T databases available via the ITU-T website at http://www.itu.int/ITU-T/ipr/.

Table of Contents

Recommendation ITU-T F.735.2

Architecture and protocols for software-defined cameras

1 Scope

Service-oriented interfaces are those APIs that are provided by the software-defined camera (SDC) operating system (OS) for the upper application developer to invoke the hardware and common software resources including raw video, algorithms inference, event, pan/tilt/zoom (PTZ), etc. to complete the intelligent analysis application development. The upper application layer can also obtain raw video, PTZ, common software and hardware resources through these APIs to complement intelligent analysis and non-intelligent business of monitoring. This Recommendation specifies the architecture and protocols for software-defined cameras, including the functional architecture of an SDC system, the service-oriented interface message protocol structure, and service-oriented interface protocols.

The scope of this recommendation includes:

1) Architecture of SDC systems;

2) SDC service-oriented interface (SOI) message protocol structure;

3) Protocols of service-oriented interface.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T F.735.1] Recommendation ITU-T F.735.1 (2020), *Requirements for software-defined cameras*.

[ITU-T F.743.1] Recommendation ITU-T F.743.1 (2015), *Requirements for intelligent visual surveillance*.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following term defined elsewhere:

3.1.1 software-defined camera [ITU-T F.735.1]: Software-defined camera is a kind of IPU (see [ITU-T F.743.1]), which provides a technical approach to decouple hardware and software and to support algorithms' on-demand deployment, online upgrade without services interrupting, continuous self-adaptive learning to adapt to various scenarios.

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

3.2.1 SDC studio: A software-defined camera (SDC) algorithm warehouse that is deployed in a cloud server and which supports service including centralized artificial intelligence model training,

intelligent algorithms online development, algorithm integration verification, application packing and release.

3.2.2 SDC controller: A client located in the front-edge of a video surveillance system, and which acts as an operation centre for massive software-defined cameras (SDCs). The main functions include SDC control (such as restart, stop, configuration, etc.) and algorithm management (such as deployment, upgrade, deletion, etc.).

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

API	Application Programming Interface
FSAAS	File System as a Service
HBTP	Hyper Binary Transfer Protocol
HTTPS	Hyper Text Transfer Protocol over Secure Socket Layer
NNIE	Neural Network Inference Engine
NoSQL	Nor only Structured Query Language
OS	Operating System
PTZ	Pan/Tilt/Zoom
SDC	Software-Defined Camera
SDK	Software Development Kit
SOI	Service-Oriented Interface
SQL	Structured Query Language
SSH	Secure Shell
VENC	Video Encoder
VI	Video Input
YUV	luminance/luma (Y) – blue luminance (U) – red luminance (V)

5 Conventions

In this Recommendation:

– The keywords "is required to" indicate a requirement which must be strictly followed and from which no deviation is permitted if conformance to this document is to be claimed.

– The keywords "is recommended" indicate a requirement which is recommended but which is not absolutely required. Thus this requirement needs not be present to claim conformance.

6 Architecture of an SDC system

The overall functional architecture of SDC is defined in [ITU-T F.735.1]. This clause describes the whole functional architecture for an SDC system and the detailed service-oriented interface architecture.

6.1 Functional architecture of an SDC system

6.1.1 Overview of entities

The SDC system functional architecture could be decomposed into three units as follows. The architecture is illustrated in Figure 6-1.

– SDC studio (SDCS)

– SDC controller (SDCC)

– SDC

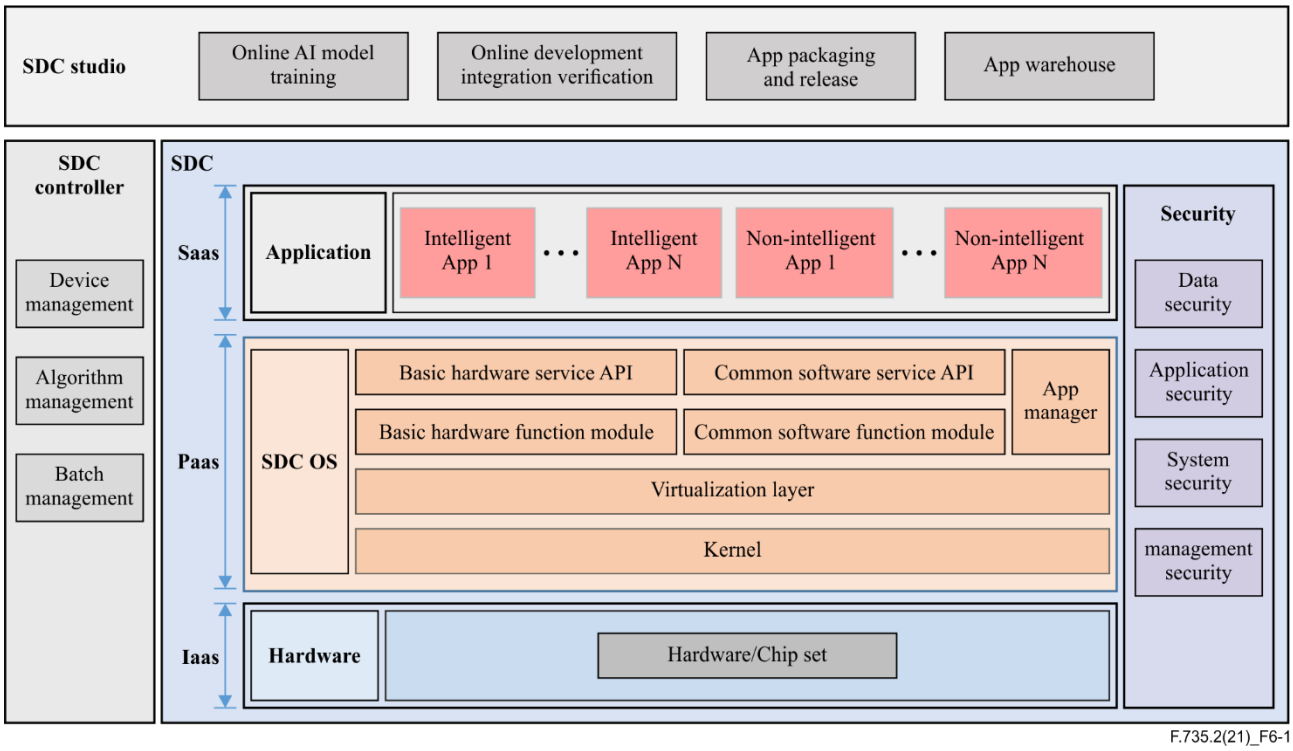

Figure 6-1 – Functional architecture of an SDC system

6.1.2 SDCS: SDC studio

SDC studio can provide the algorithm developer a self-contained and efficient development platform, and can also provide the SDC user an algorithm warehouse from where the needed algorithm can be downloaded and deployed in specific SDC devices. The functions of such platform include AI model training, verification, online development, testing, etc. The detailed functions are as follows:

– AI model training and transfer: training AI model using the specific train data and providing AI model conversion tool to complete model conversion.

– Online algorithm development: supporting online algorithm development that is based on the SDC OS service APIs.

– Algorithm integration and verification: supporting specific algorithm tested and deployed in the test environment of SDC to verify the function of this algorithm.

– Algorithm packaging and release: providing tools to complete algorithm packaging.

– Algorithm warehouse: supporting app licence management, app reviewing and app billing, etc.

6.1.3 SDCC: SDC controller

The SDC controller is a client deployed in the front-end of the surveillance system. It creates connections with SDCs and SDC studio, which acts as an operation centre for SDC management,

including camera configuration, user management, device control, and algorithm life cycle management. It is recommended that the whole management operation support batch operation. The detailed functions are as follows:

– Algorithm download: accessing the SDC studio and downloading a specific algorithm.

– Device control: video live streaming, PTZ control, image capture, recording, etc.

– User management: identification and authentication, access control, authorization, etc.

– Device configuration: protocol access parameter configuration, alarm configuration, PTZ configuration, basic video streaming configuration, etc.

– Algorithm life cycle management: algorithm deployment, activating, deactivating, upgrade and deletion in SDC.

6.1.4 SDC

The architecture of SDC is already defined in [ITU-T F.735.1], and the security function is added in Figure 6-1 to provide a full explanation of SDC. The security management contains data security, application security, system security and management security.

The security management components provide:

– Data security: data integrity verification, encrypted storage of sensitive data, system data backup, key security, etc. The data type including image, intelligent metadata (such as vehicle attribute, object detection result, the extraction feature of a human body, etc.) and alarm data in the SDC.

– Application security: installation package (APP) integrity verification, application authorization and authentication, application licence management, etc.

– System security: provision of the necessary measures to achieve system security. This includes performing authority management and control on SDC key data to prevent users from unauthorized operation, providing a digital signature and verification tool for algorithm package integrity verification and anti-hacking.

– Management security: supporting remote maintenance security (such as using hypertext transfer protocol over secure socket layer (HTTPS) and secure shell (SSH) protocol when logging into in the SDC, during sensitive data transmission, etc.) and ensuring account and password security.

For the functions of SDC hardware, SDC OS, and application layer, please refer to clause 7 in [ITU-T F.735.1].

6.2 Service-oriented interface architecture and classification

6.2.1 Service-oriented interface architecture

The software-defined camera (SDC) software architecture consists of the following layers, which are illustrated in Figure 6-2:

– Customized service software layer: Provides a variety of service functions required by end users and supports on-demand service loading.

– Common software capability service layer: Provides support for the service software layer to accelerate the production of various customization scenarios and innovative services. The common software capabilities can be loaded on demand.

– Basic hardware capability service layer: Provides interfaces for using underlying hardware resources. All basic hardware capabilities at this layer are based on the SDC hardware capability.

– SDC service-oriented interface layer: Defines the specifications of communication interfaces among capability service layers to achieve robust simplicity, high efficiency and excellent scalability.

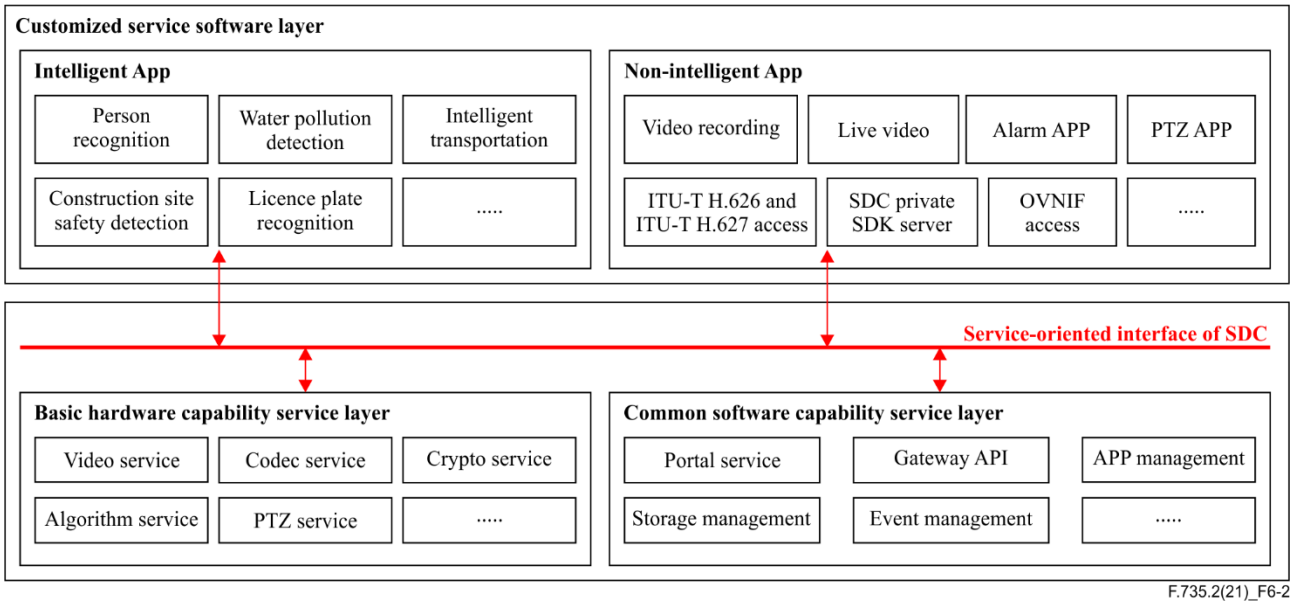

Figure 6-2 – Architecture of SDC software

6.2.2 Service-oriented interface classification

The SDC service-oriented interface can be classified into two types: the basic hardware capability service APIs and the common software capability service APIs.

Basic hardware capability service APIs

The basic hardware capability service APIs provide interfaces for using SDC hardware resources, including but not limited to getting the raw video (such as YUV frame), invoking the algorithm model inference, controlling the SDC PTZ, etc. For the function description of these service, see Table 6-1.

NOTE – YUV is abbreviation for "Luminance (Luma) / blue luminance (U) / red luminance (V)" that is a type of colour encoding representation historically used in analogue systems, such as colour components in SECAM and PAL colour spaces.

Table 6-1 – Basic hardware capability service interfaces

Service name	Function description
Video service	Provides the function of subscribing to intelligent algorithm input data – video stream data
Codec service	Provides the codec function for media chips
Crypto service	Provides the encryption and decryption functions supported by the bottom-layer chip
Algorithm service	Provides the function of loading and calculating algorithm model files for AI chips
PTZ control service	Provides the pan-tilt-zoom (PTZ) control function

Common software capability service APIs

The common software capability service APIs are used to provide the common software capabilities of the camera which include API gateway interface, event management, security, log, etc. For the function description of these services, see Table 6-2.

Table 6-2 – Common software capability service interfaces

Service name	Function description
App management	Provides app lifecycle management capabilities, such as app installation, uninstallation, enabling, disabling, starting, stopping, and watchdog.
Event management	Provides flexible event subscription mechanisms, and implements common capabilities of releasing and subscribing to event data based on the zero-copy mechanism.
Storage management	Provides the functions of managing disk partitions and storing NoSQL and SQL statements.
Portal service	Provides a unified entry for web pages of all SDC apps and the SDC OS. The entry is a human-machine interface.
Gateway API	Provides a unified entry for northbound software development kits (SDKs) of all SDC apps and the SDC OS. The northbound SDK provides the service from SDC to client.

7 SDC SOI message protocol structure

7.1 SDC SOI access mechanism definition

This clause specifies the invocation mechanism of service-oriented interfaces (SOIs). This invocation are instructions for using a file operating interface that is called file system as a service (FSAAS). All the methods are illustrated in Table 7-1. A user can invoke the **open** interface to open the file corresponding to a service and obtain the read/write handle. Then user can invoke the **read/write** interface to interact with the service. The operation for invoking the **open** interface is similar to that for setting up a TCP socket link. All the service files are located in the "/mnt/srvfs" directory in the camera system, and the service name is the file name. For example, the raw video service provided by the basic hardware capability service layer is located in the "/mnt/srvfs/video.iaas.sdc" file, this file directory address is the service URI of this raw video service interface.

Table 7-1 – The invocation methods of SOI

Methods	Description
Open	Get service handle
Read	Get service response
Write	Execute operation of interface
Close	Close service handle

The example of invoking the video service API which contains the raw video (YUV frame data) registration and subscription is shown in Figure 7-1, the variable "serviceHandle" is the video service handle, while the methods "open, write, read and close" are used to operate the service handle like file system operation. This flow is the invocation mechanism of SDC service-oriented interface.

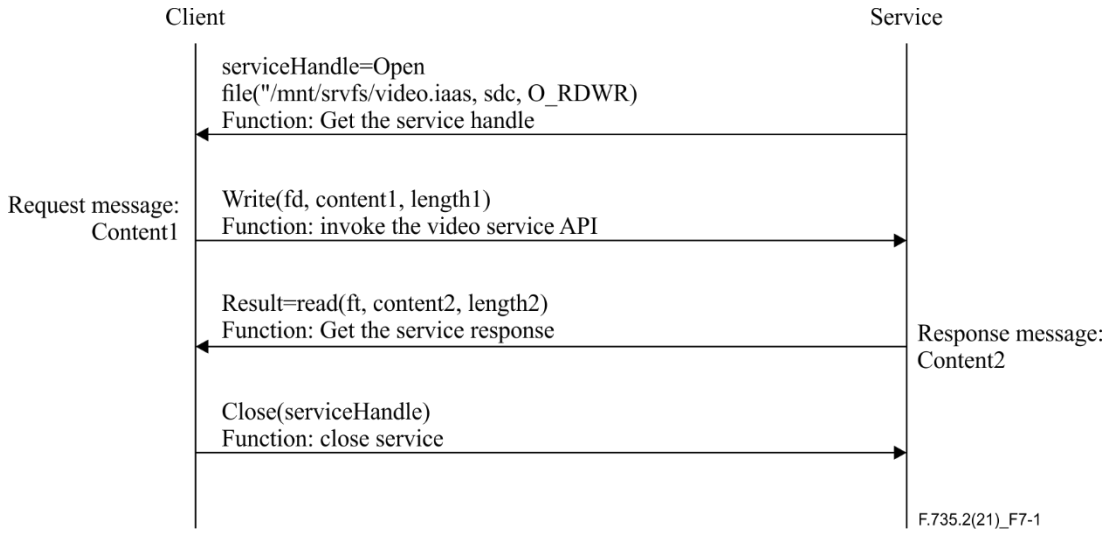

Figure 7-1 – Procedure flow of raw video service invocation

7.2 Message specification (HBTP)

This clause defines the message specification named hyper binary transfer protocol (HBTP) of such SDC SOI, which is shown in Figure 7-2. A message contains a maximum of three parts: common header, extension header and message content. The extension header and message content are optional. For details about the message content and extension header of each service interface, see the corresponding interface definition subclause in clause 8.

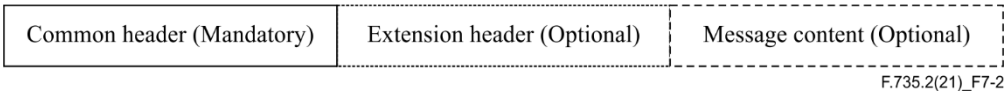

Figure 7-2 – HBTP of SDC SOI

NOTE – The message structures of the following service-oriented interfaces are defined according to the network order (high bytes stored in lower memory locations).

Common header

The common header is defined according to the network order (big endian) as shown in Figure 7-3.

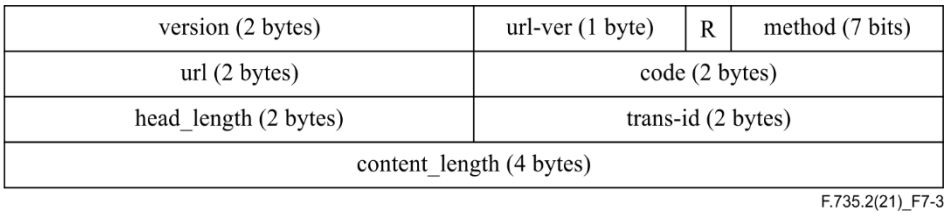

Figure 7-3 – Common header definition of SDC SOI

The definition of common header fields is shown in Table 7-2. The data structure can be defined based on the endian macro definition of the GCC; for an example, please refer to clause I.1.1.

Table 7-2 – Common header field definition

Field	Length	Semantics
version	2 bytes	Protocol version. Currently, the field has the fixed value of **0x5331**.
url_ver	1 byte	Interface version compatibility.
R	1 bit	Indicates whether a message is a request or response. The value **1** indicates response.
method	7 bits	Method definition. **1**: CREATE **2**: GET **3**: UPDATE **4**: DELETE
url	2 bytes	Resource ID
code	2 bytes	Response code (valid only when **R** is set to **1**)
head_length	2 bytes	Total length of the common header and extension header
trans-id	2 bytes	Transaction ID. The **trans_id** field in the service response header corresponds to the **trans_id** field in the request. In the pipeline scenario similar to HTTP, the server cannot ensure that the sequence of the response and request is consistent. The client can match the request and the response based on this field.
content_length	4 bytes	Total length of the message content

Extension header

The specific type of the extension header and the definition of the data structure corresponding to this type of extension header are determined by each service interface defined in clause 8. The total length is aligned by 8 bytes. The extension header definition is illustrated in Figure 7-4.

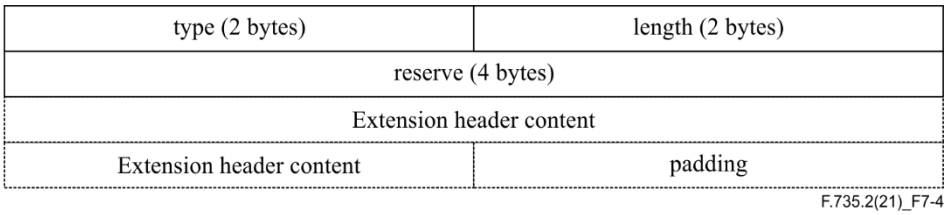

Figure 7-4 – Extension header definition of SDC SOI

The definition of extension header fields is shown in Table 7-3. The data structure can be defined based on the endian macro definition of the GCC, for the example please refer to clause I.1.2.

Table 7-3 – Extension header field definition

Field	Length	Semantics
type	2	The specific type is defined by each service
length	2	Total length of the extension header and its content, excluding the padding character.
Extension header content	hdr_len - 8	Actual message body content carried
padding	((hdr_len + 7) & ~7) – hdr_len	Ensure that the data of the entire extension header is 8-byte aligned

7.3 SOI message description format

The SOI message description is illustrated in Table 7-4. The field URI is the "url-ver" field in common header, the detailed definition of common header and extension can refer to clause 7.2. The methods CREATE, GET, UPDATE and DELETE are used in common header to complete the corresponding operation to SOI services. The message content is the specific message as defined in clause 8.

Table 7-4 – SOI message description format

Function	The interface function description				
URI	**URI Description**			**URI Value**	
Request				**Response**	
Common header		**Extension header**	**Content**	**Extension header**	**Content**
Method					
Response code					

8 Protocols of service-oriented interface

This clause defines the video service APIs and the algorithm APIs which are essential APIs for algorithm developer to get raw video frame data and execute algorithm inference. With the algorithm inference APIs, the developer can invoke the deep learning inference ability of the AI chip and realize neural network model inference.

8.1 Video service API

8.1.1 Video service function definition

The processing of the raw video which is generated from a sensor is illustrated in Figure 8-1. The raw video is changed to a YUV frame after the processing in the video process function model, and after encoding the video data is converted to ENC frame data. The video service is aimed to support the YUV or video encoder (VENC) video data subscription capability. This clause specifies the YUV and VENC video data subscription, release function and image capture function.

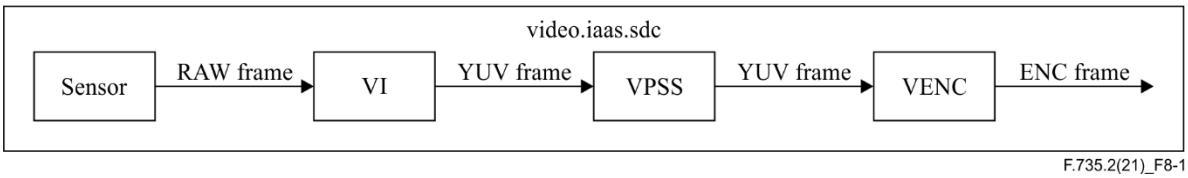

Figure 8-1 – Raw video processing procedure in SDC

8.1.2 Video logical channel definition

There are two types of channels. One is the YUV video frame data channel that is not compressed. The other is the VENC frame (H.264/H.265) channel. Users can query the attributes and usage status of each channel and set the data output format depending on their needs. If there are multiple users, the users need to collaborate with each other to eliminate potential channel resource and configuration conflicts.

8.1.2.1 YUV logical channel definition

For the definition and channel number value of YUV logical channel, see Table 8-1.

Table 8-1 – YUV logical channel definition

Resource	Type	Channel number value range
YUV snapshot frame channel	uint32	0
YUV frame data channel	uint32	1–99
NOTE – The channel number is obtained through the YUV logical channel attribute query interface. The number of channels supported by different hardware devices varies.		

8.1.2.2 VENC logical channel definition

For the definition and channel number value of VENC logical channel, see Table 8-2.

Table 8-2 – VENC logical channel definition

Resource	Type	Channel number value range
VENC frame data channel	uint32	100–104
NOTE – The channel number is obtained through the YUV logical channel attribute query interface. The number of channels supported by different hardware devices varies.		

8.1.3 YUV logical channel attribute setting

8.1.3.1 YUV logical channel attribute setting

Table 8-3 – YUV logical channel attribute setting interface definition

Function	YUV logical channel attribute setting				
URI	**URI Description**			**URI Value**	
	SDC/URL/YUV/CHANNEL			0	
Request				**Response**	
Common header		**Extension header**	**Content**	**Extension header**	**Content**
Method	UPDATE	None	sdc_yuv_channel_param	None	None
Response code	If the function is normal, response code 200 is returned. If an input error occurs, response code 400 is returned. If a server error occurs, response code 500 is returned.				

The YUV logical channel attribute setting interface definition is illustrated in Table 8-3.

8.1.3.2 Content definition

8.1.3.2.1 sdc_yuv_channel_param

The sdc_yuv_channel_param data structure definition is illustrated in Table 8-4.

Table 8-4 – sdc_yuv_channel_param definition

Element name	Type	Description
channel_number	uint32	logical channel number
Width	uint32	resolution
height	uint32	resolution
Fps	uint32	Frame per second

Table 8-4 – sdc_yuv_channel_param definition

Element name	Type	Description
on_off	uint32	The value 0 indicates that the channel is disabled. Other values indicate that the channel is enabled.
format	uint32	NV12

8.1.3.3 Extension header definition

None.

8.1.3.4 Implementation example

Refer to clause I.2.1.

8.1.4 YUV logical channel attribute query

8.1.4.1 YUV logical channel attribute query

Table 8-5 – YUV logical channel attribute query interface definition

Function	YUV logical channel attribute query				
URI	**URI Description**		**URI Value**		
	SDC/URL/YUV/CHANNEL		0		
Request				**Response**	
Common header		**Extension header**	**Content**	**Extension header**	**Content**
Method	GET	None	channel_number	None	sdc_yuv_channel_info
Response code	If the function is normal, response code 200 is returned. If an input error occurs, response code 400 is returned. If a server error occurs, response code 500 is returned.				

The YUV logical channel attribute query interface definition is illustrated in Table 8-5.

8.1.4.2 Content definition

The data structure definition of "channel_number" is shown in Table 8-6. For the "sdc_resolution" definition, please see Table 8-7, for "sdc_yuv_channel_info" definition, see Table 8-8.

8.1.4.2.1 channel_number

Table 8-6 – channel_number definition

Element name	Type	Description
channel_number	uint32	logical channel number

8.1.4.2.2 sdc_resolution

Table 8-7 – sdc_resolution definition

Element name	Type	Description
width	uint32	Width of resolution
height	uint32	Height of resolution

8.1.4.2.3 sdc_yuv_channel_info

Table 8-8 – sdc_yuv_channel_info definition

Element name	Type	Description
max_resolution	sdc_resolution	Maximum resolution supported by the channel.
param	sdc_yuv_channel_param	YUV channel attribute
is_snap_channel	uint32	Indicates whether the channel is a snapshot channel. The options are as follows: 0: common channel; 1: snapshot channel.
src_id	uint32	Data source. The data source uses the IP address to identify the data source (local camera or other cameras) of the channel. For the local multi-lens camera, the data source is identified by 127.0.0.*x*, where *x* starts from 1 and is incremented by 1.
subscriber_cnt	uint32	Number of users who subscribe to data of this channel. The value greater than 1 indicates that multiple users subscribe to data of the same channel. Video data is transferred to users in zero-copy mode. If a user modifies the data obtained from this channel, other users will read the data after modification.
resolution_moditfy	uint32	Indicates whether the resolution parameter of the channel can be modified. The options are as follows: 0: no; 1: yes.

8.1.4.3 Extension header definition

None.

8.1.4.4 Implementation example

Refer to clause I.2.2.

8.1.5 YUV frame data subscription

8.1.5.1 YUV frame data subscription interface

Table 8-9 – YUV frame data subscription interface definition

<table>
<tr><td>Function</td><td colspan="5">YUV frame data subscription</td></tr>
<tr><td rowspan="2">URI</td><td colspan="3">URI Description</td><td colspan="2">URI Value</td></tr>
<tr><td colspan="3">SDC_URL_YUV_DATA</td><td colspan="2">1</td></tr>
<tr><td colspan="4">Request</td><td colspan="2">Response</td></tr>
<tr><td colspan="2">Common header</td><td>Extension header</td><td>Content</td><td>Extension header</td><td>Content</td></tr>
<tr><td>Method</td><td>GET</td><td>Refer to clause 8.1.5.3.1</td><td>channel_number (refer to clause 8.1.4.2)</td><td>Refer to clause 8.1.5.3.2</td><td>sdc_yuv_data</td></tr>
<tr><td>Response code</td><td colspan="5">If the function is normal, response code 200 is returned. If flow control is enabled, response code 509 (Bandwidth Limit Exceeded) and empty response content are returned. If an input error occurs, response code 400 is returned. If a server error occurs, response code 500 and empty response content are returned.</td></tr>
</table>

The YUV frame data subscription interface definition is illustrated in Table 8-9.

8.1.5.2 Content definition

For the "sdc_yuv_frame" definition, please see Table 8-10, for "sdc_yuv_data" definition, see Table 8-11.

8.1.5.2.1 sdc_yuv_frame

Table 8-10 – sdc_yuv_frame data

Element name	Type	Description
addr_phy	uint32	Physical address
addr_virt	uint32	Virtual address
size	uint32	Size of frame
width	uint32	Width pixel of resolution
height	uint32	Height pixel of resolution
stride	uint32	Image width rounded up, for 16 byte aligned
format	uint32	Frame format
reserve	uint32	Reserve field

8.1.5.2.2 sdc_yuv_data

Table 8-11 – sdc_yuv_data

Element name	Type	Description
Channel	uint32	Channel number
Reserve	uint32	Reserve field
Pts	uint32	Timestamp carried by the bottom-layer chip, in microseconds.
Pts_sys	uint32	Timestamp of the data obtained by the server, in microseconds.
Frame	sdc_yuv_frame	sdc_yuv_frame

8.1.5.3 Extension header

8.1.5.3.1 Extension header for request

The request extension header can be SDC_HEAD_YUV_SYNC, SDC_HEAD_YUV_CACHED_COUNT_MAX or SDC_HEAD_YUV_PARAM_MASK, the definition are as follows:

a) SDC_HEAD_YUV_SYNC: Extension header for identifying multi-channel synchronization is illustrated in Figure 8-2.

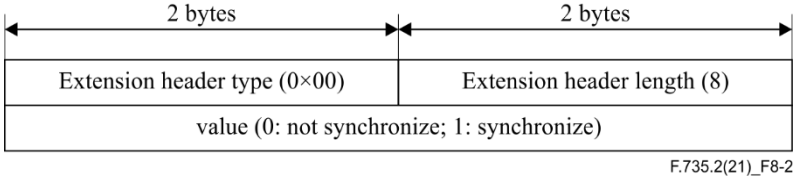

Figure 8-2 – Extension header: SDC_HEAD_YUV_SYNC

The value is saved in the Reserve field in the extension header. To subscribe to data of multiple channels, if this extension header does not exist, the synchronization policy is used by default.

b) SDC_HEAD_YUV_CACHED_COUNT_MAX: Extension header for setting the flow control threshold is illustrated in Figure 8-3.

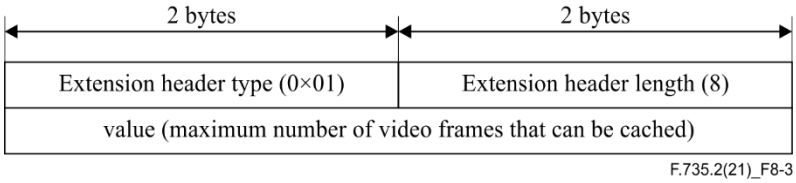

Figure 8-3 – Extension header: SDC_HEAD_YUV_CACHED_COUNT_MAX

The value is saved in the Reserve field in the extension header. If this extension header does not exist, the default number of cached video frames on the server is calculated as follows: Sum of frame rates of each channel x 2 (that is, YUV frame resources are occupied for up to 2 s).

If the total number of cached (not released) video frames exceeds the threshold, the server starts the flow control function and does not forward the subscribed video frame data. Therefore, users need to invoke the interface for releasing YUV frames as needed.

c) SDC_HEAD_YUV_PARAM_MASK: Extension header for subscribing to frame parameters is illustrated in Figure 8-4.

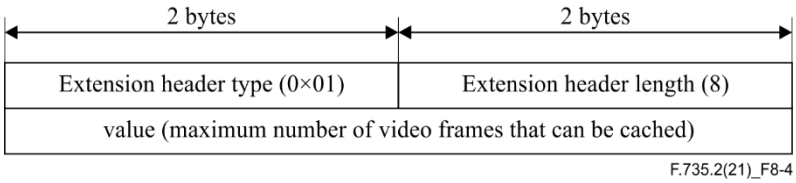

Figure 8-4 – Extension header: SDC_HEAD_YUV_PARAM_MASK

The value is saved in the Reserve field in the extension header. If this extension header does not exist, the default number of cached video frames on the server is calculated as follows: Sum of frame rates of each channel x 2 (that is, YUV frame resources are occupied for up to 2 s).

If the total number of cached (not released) video frames exceeds the threshold, the server starts the flow control function and does not forward the subscribed video frame data. Therefore, users need to invoke the interface for releasing YUV frames as needed.

8.1.5.3.2 Extension header for response

The request extension header can be SDC_HEAD_YUV_CACHED_COUNT, or SDC_HEAD_YUV_PARAM_SNAP, the definition are as follows:

a) SDC_HEAD_YUV_CACHED_COUNT: Extension header for counting cached video frames, the format definition is illustrated in Figure 8-5.

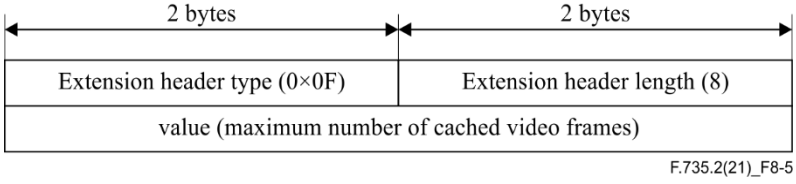

Figure 8-5 – Extension header: SDC_HEAD_YUV_CACHED_COUNT

b) SDC_HEAD_YUV_PARAM_SNAP: Extension header for snapshot frame parameters, the format definition is illustrated in Figure 8-6.

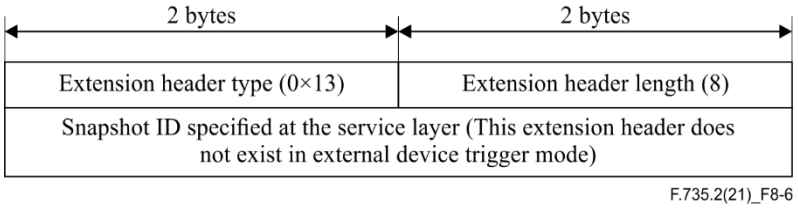

Figure 8-6 – Extension header: SDC_HEAD_YUV_PARAM_SNAP

For details about the specified snapshot ID, see the input parameters of the snapshot frame interface in clause 8.1.11.

8.1.5.4 Implementation example

Refer to YUV frame subscription example in clause I.2.3.

8.1.6 YUV frame data release

8.1.6.1 YUV frame data release interface

Table 8-12 – YUV frame data release interface definition

Function	YUV frame data release				
URI	**URI Description**		**URI Value**		
	SDC_URL_YUV_DATA		0x01		
Request				**Response**	
Common header		**Extension header**	**Content**	**Extension header**	**Content**
Method	DELETE	None	sdc_yuv_data	None	None

The YUV frame data release interface definition is illustrated in Table 8-12.

8.1.6.2 Content definition

The definition of sdc_yuv_data refer to 8.1.5.2.2.

8.1.6.3 Extension header

None.

8.1.6.4 Implementation example

Refer to the YUV frame data subscription example in clause I.2.3.

8.1.7 VENC logical channel attribute setting

8.1.7.1 VENC logical channel attribute setting interface

The VENC logical channel attribute setting interface definition is illustrated in Table 8-13.

Table 8-13 – VENC logical channel attribute setting interface definition

Function	VENC logical channel attribute setting				
URI	**URI Description**		**URI Value**		
	SDC_URL_VENC_CHANNEL		2		
Request				**Response**	
Common header		**Extension header**	**Content**	**Extension header**	**Content**
Method	UPDATE	None	sdc_venc_channel_param	None	None
Response code	If the function is normal, response code 200 is returned. If an input error occurs, response code 400 is returned. If a server error occurs, response code 500 is returned.				

8.1.7.2 Content definition

For the "sdc_venc_channel_param" definition, please see Table 8-14.

8.1.7.2.1 sdc_venc_channel_param

Table 8-14 – sdc_venc_channel_param

Element name	Type	Description
channel	uint32	The number of the logical channel
width	uint32	Resolution
height	uint32	Resolution
fps	uint32	Frame per second
on_off	uint32	The value 0 indicates that the channel is disabled. Other values indicate that the channel is enabled.
format	uint32	Compression format: H264 / H265 / MJPEG

8.1.7.3 Extension header definition

None.

8.1.7.4 Implementation example

Refer to clause I.2.4.

8.1.8 VENC logical channel attribute query

8.1.8.1 VENC logical channel attribute query interface

The VENC logical channel attribute query interface definition is illustrated in Table 8-15.

Table 8-15 – VENC logical channel attribute query interface definition

Function	VENC logical channel attribute query				
URI	**URI Description**		**URI Value**		
	SDC_URL_VENC_CHANNEL		2		
Request				**Response**	
Common header		**Extension header**	**Content**	**Extension header**	**Content**
Method	GET	None	channel_number	None	None
Response code	If the function is normal, response code 200 is returned. If an input error occurs, response code 400 is returned. If a server error occurs, response code 500 is returned.				

8.1.8.2 Content definition

The definition of "channel_number" refer to clause 8.1.4.2.

8.1.8.3 Extension header definition

None.

8.1.8.4 Implementation example

Refer to clause I.2.5.

8.1.9 VENC frame data subscription

8.1.9.1 VENC frame data subscription API

The VENC frame data subscription interface definition is illustrated in Table 8-16.

Table 8-16 – VENC frame data subscription interface definition

Function	VENC frame data subscription				
URI	**URI Description**		**URI Value**		
	SDC_URL_VENC_DATA		0x03		
Request				**Response**	
Common header		**Extension header**	**Content**	**Extension header**	**Content**
Method	GET	Refer to clause 8.1.9.3.1	channel_number (refer to clause 8.1.4.2)	Refer to clause 8.1.9.3.2	sdc_venc_data
Response Code	If the function is normal, response code 200 is returned. If flow control is enabled, response code 509 (Bandwidth Limit Exceeded) and empty response content are returned. If an input error occurs, response code 400 is returned. If a server error occurs, response code 500 and empty response content are returned.				

8.1.9.2 Content definition

For the definition of "sdc_venc_frame", please see Table 8-17. For the definition of "sdc_venc_data", please see Table 8-18.

8.1.9.2.1 sdc_venc_frame

Table 8-17 – sdc_venc_frame

Element name	Type	Description
addr_phy	uint32	Physical address
addr_virt	uint32	Virtual address
size	uint32	The size of frame
width	uint32	Resolution width
height	uint32	Resolution height
format	uint32	SDC_H264 OR SDC_H265
frame_type	uint32	SDC_VENC_FRAME_I/P/B

8.1.9.2.2 sdc_venc_data

Table 8-18 – sdc_venc_data

Element name	Type	Description
channel	uint32	The number of the logical channel
reserve	uint32	Reserve
frame_pts	uint64	Frame timestamp
pts_sys	uint64	System timestamp
frame	sdc_venc_frame	sdc_venc_frame

8.1.9.3 Extension header

8.1.9.3.1 Extension header for request

SDC_HEAD_VENC_CACHED_COUNT_MAX: Extension header for setting the flow control threshold; the format definition is illustrated in Figure 8-7.

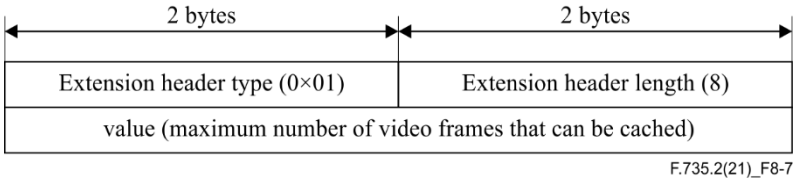

Figure 8-7 – Extension header: SDC_HEAD_VENC_CACHED_COUNT_MAX

The value is saved in the **Reserve** field in the extension header. If this extension header does not exist, the default number of cached video frames on the server is calculated as follows: Sum of frame rates of each channel x 2. (That is, VENC frame resources are occupied for up to 2s. This number is not related to the size of each frame. The client caches consecutive video frames.)

If the total number of cached (not released) video frames exceeds the threshold, the server starts the flow control function and does not forward the subscribed video frame data. Therefore, users need to invoke the interface for releasing VENC frames as needed.

8.1.9.3.2 Extension header for response

SDC_HEAD_VENC_CACHED_COUNT: Extension header for counting cached video frames, the format definition is illustrated in Figure 8-8.

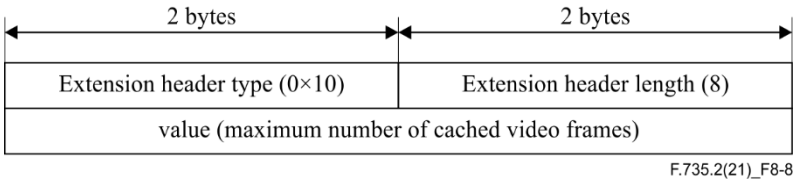

Figure 8-8 – Extension header: SDC_HEAD_VENC_CACHED_COUNT

8.1.9.4 Implementation example

Refer to clause I.2.6.

8.1.10 VENC frame data release

8.1.10.1 VENC frame data release interface

The VENC frame data release interface definition is illustrated in Table 8-19.

Table 8-19 – VENC frame data release interface definition

Function	VENC frame data release				
URI	**URI Description**		**URI Value**		
	SDC_URL_VENC_DATA		0x03		
Request				**Response**	
Common header		**Extension header**	**Content**	**Extension header**	**Content**
Method	DELETE	None	sdc_venc_data (refer to clause 8.1.9.2.2)	None	None

8.1.10.2 Content definition

None.

8.1.10.3 Extension header definition

None.

8.1.10.4 Implementation example

For an implementation example of the interface please refer to the VENC frame data subscription interface in clause I.2.6.

8.1.11 Snapshot interface

8.1.11.1 Snapshot interface

The snapshot interface definition is illustrated in Table 8-20.

Table 8-20 – Snapshot interface definition

Function	Snapshot interface				
URI	**URI Description**		**URI Value**		
	SDC_URL_YUV_SNAP		0x04		
Request				**Response**	
Common header		**Extension header**	**Content**	**Extension header**	**Content**
Method	CREATE	None	sdc_yuv_snap_param	None	None
Response code	If the function is normal, response code 200 is returned. Other values are error codes.				

Snapshot frames are classified into two types: snapshot proactively triggered by external devices and snapshot proactively triggered by the service layer. To obtain the two types of snapshot frame data, the service layer needs to subscribe to data of the snapshot channel. During subscription, you need to set the extension header to obtain the snapshot frame parameters.

NOTE – The snapshot request must be sent on the handle of the subscribed snapshot channel. Otherwise, the snapshot frame data cannot be received.

8.1.11.2 Content definition

For the definition of "sdc_yuv_snap_param", please see Table 8-21.

8.1.11.2.1 sdc_yuv_snap_param

Table 8-21 – sdc_yuv_snap_param

Element name	Type	Description
id	uint32	User id. When a user subscribes to snapshot frame data, the user can obtain the corresponding ID for matching.
num	uint32	Number of snapshots
frame_pts	uint64	Frame timestamp
interval_msec	uint64	Snapshot interval, in milliseconds. A maximum of four snapshot intervals can be set. The number of snapshot intervals equals the number of snapshots minus 1. Currently, a maximum of three snapshots are supported. That is, a maximum of two snapshot intervals can be set currently.
portaddr_name	char[16]	Port name

8.1.11.3 Extension header

None.

8.1.11.4 Implementation example

Refer to the YUV frame data subscription example in clause I.2.3.

8.2 Algorithm inference API

This clause defines the algorithm inference API based on the neural network inference engine (NNIE) model, NNIE is a widely used acceleration engine in surveillance system Soc.

8.2.1 Model creation

8.2.1.1 Model creation interface

This interface is used to create executable models based on the input WK file. The NNIE model creation interface definition is illustrated in Table 8-22.

Table 8-22 – Model creation interface definition

Function	NNIE model creation				
URL	**URI Description**		**URI Value**		
	SDC_URL_NNIE_MODEL		0		
Request				**Response**	
Common header		**Extension header**	**Content**	**Extension header**	**Content**
Method	CREATE	Refer to clause 8.2.1.3	Refer to clause 8.2.1.2	None	Model
Response code	If the function is normal, response code 200 is returned. Other values are error codes.				

8.2.1.2 Content definition

If the extension header SDC_HEAD_NNIE_MODEL_CONTENT_TYPE is set to 0 or not defined, the request content is SDCMmz.

If the extension header SDC_HEAD_NNIE_MODEL_CONTENT_TYPE is set to 1, the content specifies the file name in the following format: char filename[].hbtp.content_length indicates the length of the file name. Therefore, whether the file name ends with **NULL** does not affect the function.

8.2.1.2.1 SDCMmz

For the definition of "SDCMmz", please see Table 8-23.

Table 8-23 – SDCMmz

Element name	Type	Description
addr_phy	uint64	Physical address
addr_virt	uint64	Virtual address
size	uint32	size
cookie	uint32 [4]	Cookie text for the creation session

8.2.1.3 Extension header

8.2.1.3.1 Request extension header (SDC_HEAD_DECODED_YUV_ACCEPT_TYPE)

The extension header definition for request is shown Figure 8-9. If no extension header is defined, the MMZ mode is used.

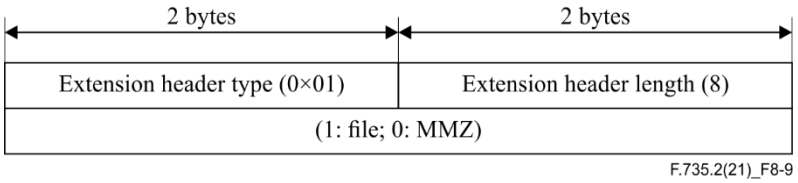

Figure 8-9 – Extension header: SDC_HEAD_DECODED_YUV_ACCEPT_TYPE

8.2.1.4 Implementation example

The reference implementation example of interface please refer to clause I.3.1.

8.2.2 Model deletion

8.2.2.1 Model deletion interface

This interface is used to release the executable model created based on the WK file. The NNIE model deletion interface definition is illustrated in Table 8-24.

Table 8-24 – NNIE model deletion interface definition

Function	NNIE model deletion				
URL	**URI Description**		**URI Value**		
	SDC_URL_NNIE_MODEL		0		
Request				**Response**	
Common header		**Extension header**	**Content**	**Extension header**	**Content**
Method	DELETE	None	SVP_NNIE_MODEL_S	None	None
Response code	No response is returned in resource release interfaces				

8.2.2.2 Content definition

The "SVP_NNIE_MODEL_S" is a defined structure type base on NNIE mode engine which is widely used in video surveillance system. The definition of "SVP_NNIE_MODEL_S" is out scope of this Recommendation.

8.2.2.3 Extension header

None.

8.2.2.4 Implementation example

The implementation example of interface please refer to clause I.3.2.

8.2.3 Model forward

8.2.3.1 Model forward interface

This interface is used for multi-node input and output CNN network prediction. The NNIE model forward interface definition is illustrated in Table 8-25.

NOTE – This interface is equivalent to combining "HI_MPI_SVP_NNIE_GetTskBufSize", "HI_MPI_SVP_NNIE_Forward" and "HI_MPI_SVP_NNIE_Query" interfaces, and the latter is the response, the definition of HI_MPI_SVP_NNIE_GetTskBufSize, HI_MPI_SVP_NNIE_Forward and HI_MPI_SVP_NNIE_Query can refer to the NNIE engine, and is out scope of this Recommendation.

Table 8-25 – NNIE forward interface definition

Function	NNIE forward				
URL	**URI Description**		**URI Value**		
	SDC_URL_NNIE_FORWARD		1		
Request				**Response**	
Common header		**Extension header**	**Content**	**Extension header**	**Content**
Method	GET	Refer to clause 8.2.3.3	Refer to clause 8.2.3.2	Refer to clause 8.2.3.3	None
Response code	If the operation succeeds, response code 200 is returned. Other values are error codes. If flow control is enabled, response code 509 (Bandwidth Limit Exceeded) and empty response content are returned.				

8.2.3.2 Content definition

For the definition of "sdc_nnie_forward_ctrl", please see Table 8-26. For the definition of "sdc_nnie_forward_para", please see Table 8-27.

8.2.3.2.1 sdc_nnie_forward_ctrl

Table 8-26 – sdc_nnie_forward_ctrl

Element name	Type	Description
netseg_id	uint64	Physical address
max_batch_num	uint64	Virtual address
max_bbox_num	uint32	size

8.2.3.2.2 sdc_nnie_forward_para

Table 8-27 – sdc_nnie_forward_para

Element name	Type	Description
model	SVP_NNIE_MODEL_S	/
foward_ctl	sdc_nnie_forward_ctrl	/
astSrc	SVP_SRC_BLOB_S[16]	/
astDst	SVP_DST_BLOB_S[16]	/

NOTE – The definition of SVP_NNIE_MODEL_S and SVP_SRC_BLOB_S are out scope of this Recommendation.

8.2.3.3 Extension header

8.2.3.3.1 Extension header for request

a) Task Priority (SDC_HEAD_PRI), the header definition is shown in Figure 8-10.

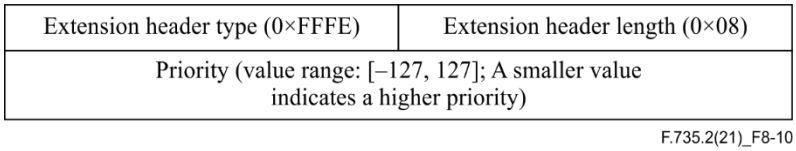

Figure 8-10 – Extension header: SDC_HEAD_PRI

A smaller value indicates a higher priority. The maximum waiting delay (controlled by the server, 300 ms by default) is configured for a low-priority task. If the maximum waiting delay is exceeded, a high-priority task cannot pre-empt resources of the low-priority task.

8.2.3.4 Implementation example

The implementation example, please refer to clause I.3.3.

Appendix I

Data Type Definition and API Example

(This appendix does not form an integral part of this Recommendation.)

I.1 SOI data type definition example

I.1.1 Common header definition

The common header data structure can be defined as follows based on the endian macro definition of GCC:

```
struct sdc_common_head
{
uint16_t        hbtp_ver;
uint8_t         uri_ver;
#if defined(__BYTE_ORDER__) && defined(__ORDER_LITTLE_ENDIAN__) &&
defined(__ORDER_BIG_ENDIAN__)
#if (__BYTE_ORDER__ == __ORDER_LITTLE_ENDIAN__)
uint8_t         method: 7;
uint8_t         response: 1;
#elif (__BYTE_ORDER__ == __ORDER_BIG_ENDIAN__)
uint8_t         response: 1;
uint8_t         method: 7;
#else
#error "unknown __BYTE_ORDER__"
#endif
#else
#error "don't define __BYTE_ORDER__ or __ORDER_LITTLE_ENDIAN__ or
__ORDER_BIG_ENDIAN__"
#endif
uint16_t        uri;
uint16_t        code;
uint16_t        head_length;
uint16_t        trans_id;
uint32_t        content_length;
};
The response code is valid only when R is set to 1. The following describes
the response codes:
HBTP_CODE_200 = 200,  // OK
HBTP_CODE_400 = 400,  // Bad Request
HBTP_CODE_401 = 401,  // Unauthorized
HBTP_CODE_403 = 403,  // Forbidden
HBTP_CODE_404 = 404,  // Not Found
HBTP_CODE_500 = 500,  // Internal Server Error
HBTP_CODE_509 = 509 and // Flow control
```

I.1.2 Extension header definition

```
struct sdc_extend_head
{
uint16_t hdr_type;
uint16_t hdr_len;
uint32_t reserve;
};
```

I.2 Video service API example

I.2.1 YUV logical channel attribute settings example

Content definition

```
#define SDC_YVU_420SP0 // YUV frame. Currently, only YVU_420SP is supported.
struct sdc_yuv_channel_param
{
uint32_t channel;// The value 0 indicates the snapshot channel (supported only
for the ITS model).
uint32_t width;
uint32_t height;
uint32_t fps;     // The value 0 indicates the default frame rate at the
bottom layer.
uint32_t on_off;  // The value 0 indicates that the channel is disabled. Other
values indicate that the channel is enabled.
uint32_t format;  //SDC_YVU_420SP
};
Batch configuration is supported. The number of channels is calculated as
follows: hbtp.content_length/sizeof(sdc_yuv_channel_param).
If the input parameter settings are correct, the settings take effect. Channel
conflicts between services are resolved at the service layer.
```

Reference example

```
#include "sdc.h"
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/uio.h>
int main(int argc,char* argv[])
{
struct sdc_yuv_channel_param param = {
.channel = 1,
.width = 1280,
.height = 720,
.fps = 25,
.on_off = 1,
.format = SDC_YVU_420SP,
};
struct sdc_common_head head = {
.version = SDC_VERSION, //0x5331
.url = SDC_URL_YUV_CHANNEL, //0x00
.method = SDC_METHOD_UPDATE, //0x02
.content_length = sizeof(param),
.head_length = sizeof(head),
};
struct iovec iov[2] = { {.iov_base = &head, .iov_len = sizeof(head)},
{.iov_base = &param, .iov_len = sizeof(param) }};
int nret;
int fd = open("/mnt/srvfs/video.iaas.sdc", O_RDWR);
if(fd < 0) goto fail;
nret = writev(fd, iov, 2);
if(nret < 0) goto fail;
nret = read(fd,&head, sizeof(head));
if(head.code == SDC_CODE_200 /** 200 */) {
//...
}else{
//...
}
close(fd);
return 0;
fail:
exit(1); //fd will be closed after exit
}
```

I.2.2 YUV logical channel attribute querying example

Content definition

```
uint32_t channel;
/**Attributes of logical channels can be queried in batches. The number of
channels is calculated as follows: hbtp.content_length/sizeof(uint32_t).
If there is no content, that is, the value of hbtp.content_length is 0, the
attributes of all channels are returned.**/
struct sdc_resolution {
uint32_t width;
uint32_t height;
};
struct sdc_yuv_channel_info
{
struct sdc_yuv_channel_param param;
struct sdc_resolution max_resolution; // Maximum resolution supported by the
channel.
uint32_t is_snap_channel; // Indicates whether the channel is a snapshot
channel. The options are as follows: 0: common channel; 1: snapshot channel.
uint32_t src_id;         // Data source. The data source uses the IP address
to identify the data source (local camera or other cameras) of the channel.
For the local multi-lens camera, the data source is identified by 127.0.0.x,
where x starts from 1 and is incremented by 1.
uint32_t subscriber_cnt; // Number of users who subscribe to data of this
channel. The value greater than 1 indicates that multiple users subscribe to
data of the same channel. Video data is transferred to users in zero-copy
mode. If a user modifies the data obtained from this channel, other users will
read the data after modification.
uint32_t resolution_moditfy;   // Indicates whether the resolution parameter
of the channel can be modified. The options are as follows: 0: no; 1: yes.
};
/**When attributes of logical channels are queried in batches, the response
content is also returned in batches. The number of channels returned in
batches is calculated as follows: hbtp.content_length/sizeof(struct
sdc_yuv_channel_info).
**/
```

Reference example

```
#include "sdc.h"
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
static void display_yuv_channel_info(struct sdc_yuv_channel_info* info);
int main(int argc,char* argv[])
{
int nret,i;
char buf[1024] = { 0 };
struct sdc_yuv_channel_info* info;
struct sdc_common_head* head = (struct sdc_common_head*)buf;
/** query all channels' info */
head->version = SDC_VERSION; //0x5331
head->url = SDC_URL_YUV_CHANNEL; //0x00
head->method = SDC_METHOD_GET; //0x02
head->head_length = sizeof(*head);
int fd = open("/mnt/srvfs/video.iaas.sdc", O_RDWR);
if(fd < 0) goto fail;
nret = write(fd, head, head->head_length + head->content_length);
if(nret < 0) goto fail;
nret = read(fd,buf,sizeof(buf));
if(nret < 0 || head->code != SDC_CODE_200) goto fail;
info = (struct sdc_yuv_channel_info*)&buf[head->head_length];
for(i = 0; i < head->content_length / sizeof(*info); ++i, ++info) {
```

```
display_yuv_channel_info(info);
}
close(fd);
return 0;
fail:
exit(1); //fd will be closed after exit
}
static void display_yuv_channel_info(struct sdc_yuv_channel_info* info)
{
}
```

I.2.3 YUV frame data subscription example

Content definition

```
uint32_t channel;
Multiple channels can be obtained. The number of channels is calculated as
follows: hbtp.content_length/sizeof(uint32_t).
/**After the subscription is successful, multiple packets are received
consecutively. The first response packet has no content. **/
/**The subsequent packets include YUV frame data. The structure is defined as
follows: **/
struct sdc_yuv_frame
{
uint64_t   addr_phy;
uint64_t   addr_virt; // Cache mapping is implemented by default.
uint32_t   size;
uint32_t    width;
uint32_t    height;
uint32_t   stride;
uint32_t   format;
uint32_t    reserve;
uint32_tcookie[4]; // Required by service functions or during commissioning.
}
struct sdc_yuv_data
{
uint32_t channel;
uint32_t reserve;
uint64_t pts; // Timestamp carried by the bottom-layer chip, in microseconds.
uint64_t pts_sys; // Timestamp of the data obtained by the server, in
microseconds.
struct sdc_yuv_frame frame;
};
/**When channel data is subscribed in batches, the number of channels is
calculated as follows: hbtp.content_length/sizeof(struct sdc_yuv_data).
The bottom-layer chip time of multi-lens SDCs may be inconsistent. The
synchronization of multiple channels depends on the system time, that is, the
pts_sys field rather than the pts field.
Note: The trans_id field in the response headers of multiple packets is the
same as the trans_id field in the subscription request.**/
```

Reference example

```
#include "sdc.h"
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
static void display_extend_head(struct sdc_extend_head* extend_head) {}
static void display_yuv_data(struct sdc_yuv_data* yuv_data) {}
int main(int argc,char* argv[])
{
int nret,i,fd;
char buf[1024] = { 0 };
struct sdc_common_head* head = (struct sdc_common_head*) buf;
struct sdc_extend_head* extend_head;
```

```
struct sdc_yuv_data* yuv_data;
struct sdc_yuv_snap_param* snap_param;
uint32_t* channel;
fd = open("/mnt/srvfs/video.iaas.sdc", O_RDWR);
if(fd < 0) goto fail;
head->version = SDC_VERSION;
head->url = SDC_URL_YUV_DATA;
head->method = SDC_METHOD_GET;
head->head_length = sizeof(*head);
/** The maximum number of cached YUV frames is 10. */
extend_head = (struct sdc_extend_head*)&buf[head->head_length];
extend_head->type = SDC_HEAD_YUV_CACHED_COUNT_MAX;
extend_head->length = sizeof(*extend_head);
extend_head->reserve = 10;
/**
* #define sdc_extend_head_length(extend_head) (((extend_head)->length + 7) &
~7)
* The length of the extension header defined in the existing SDC service-
oriented interface is aligned by 8 bytes. Developers also need to write "head-
>head_length += extend_head->length" to ensure that the length of the
extension header is 8-byte aligned.
*/
head->head_length += sdc_extend_head_length(extend_head);
/** Subscribe to data of two channels. No synchronization is required because
one channel is the snapshot channel. */
extend_head = (struct sdc_extend_head*)&buf[head->head_length];
extend_head->type = SDC_HEAD_YUV_SYNC;
extend_head->length = sizeof(*extend_head);
extend_head->reserve = 0;
head->head_length += sdc_extend_head_length(extend_head);
/** Subscribe to data of two channels. */
channel = (uint32_t*)&buf[head->head_length];
 channel[0] = 0; // The value 0 indicates the snapshot channel by default. The
snapshot channel ID can be obtained through the query interface.
channel[1] = 1;
head->content_length = 2 * sizeof(channel[0]);
nret = write(fd, head, head->head_length + head->content_length);
if(nret < 0) goto fail;
for(;;) {
nret = read(fd, buf,sizeof(buf));
if(nret < 0) goto fail;
switch(head->url){
case SDC_URL_YUV_SNAP:
/** Response to snapshot processing. */
if(head->code != SDC_CODE_200) {
// log error info
}
continue;
case SDC_URL_YUV_DATA:
break;
default:
continue;
}
/**
*#define sdc_extend_head_next(extend_head) ((struct
sdc_extend_head*)((char*)extend_head + sdc_extend_head_length(extend_head)))
* #define sdc_extend_head_first(common_head) ((struct
sdc_extend_head*)(common_head + 1))
* #define sdc_for_each_extend_head(common_head, extend_head) \
* for( extend_head = sdc_extend_head_first(common_head); (char*)extend_head -
(char*)common_head < common_head->head_length; extend_head =
sdc_extend_head_next(extend_head))
*/
sdc_for_each_extend_head(head, extend_head) {
```

```
display_extend_head(extend_head);
}
for(i = 0, yuv_data = (struct sdc_yuv_data*)&buf[head->head_length]; i < head-
>content_length / sizeof(*yuv_data); ++i, ++yuv_data) {
display_yuv_data(yuv_data);
}
/** free yuv_data */
head->response = head->code = 0;
head->method = SDC_METHOD_DELETE;
(void)write(fd,head,head->head_length + head->content_length); // server
ignore extended headers
/** Trigger the snapshot taking action. */
if( 1 ) {
head->url = SDC_URL_YUV_SNAP;
head->method = SDC_METHOD_CREATE;
head->head_length = sizeof(*head);
head->content_length = sizeof(*snap_param);
snap_param = (struct sdc_yuv_snap_param*)&buf[head->head_length];
snap_param->id = 100;
snap_param->num = 1;
snap_param->interval_msec = 0;
nret = write(fd, head, head->head_length + head->content_length);
if(nret < 0) goto fail;
/** Data read immediately may not be the snapshot response data. */
}
}
return 0;
fail:
exit(1);
}
```

I.2.4 VENC logical channel attribute setting example

Content definition

```
#define SDC_H264            0
#define SDC_H265            1
#define SDC_MJPEG          2
struct sdc_venc_channel_param
{
uint32_t channel;
uint32_t width;
uint32_t height;
uint32_t fps;     // The value ranges from 1 to the value of MaxFps.
uint32_t on_off;  // The value 0 indicates that the channel is disabled. Other
values indicate that the channel is enabled.
uint32_t format;  // SDC_H264 \ SDC_H265\ SDC_MJPEG
};
/**Batch configuration is supported. The number of channels is calculated as
follows: hbtp.content_length/sizeof(sdc_venc_channel_param).
If the input parameter settings are correct, the settings take effect. Channel
conflicts between services are resolved at the service layer.**/
```

Reference example

```
#include "sdc.h"
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/uio.h>
int main(int argc,char* argv[])
{
struct sdc_yuv_channel_param param = {
.channel = 100,
```

```
.width = 1280,
.height = 720,
.fps = 25,
.on_off = 1,
.format = SDC_H264,
};
struct sdc_common_head head = {
.version = SDC_VERSION, //0x5331
.url = SDC_URL_VENC_CHANNEL, //0x02
.method = SDC_METHOD_UPDATE, //0x02
.content_length = sizeof(param),
.head_length = sizeof(head),
};
struct iovec iov[2] = { {.iov_base = &head, .iov_len = sizeof(head)},
{.iov_base = &param, .iov_len = sizeof(param) }};
int nret;
int fd = open("/mnt/srvfs/video.iaas.sdc", O_RDWR);
if(fd < 0) goto fail;
nret = writev(fd, iov, 2);
if(nret < 0) goto fail;
nret = read(fd,&head, sizeof(head));
if(head.code == SDC_CODE_200 /** 200 */) {
//...
}else{
//...
}
close(fd);
return 0;
fail:
exit(1); //fd will be closed after exit
}
```

I.2.5 VENC logical channel attribute querying example

Content definition

```
uint32_t channel;
/**Attributes of logical channels can be queried in batches. The number of
channels is calculated as follows: hbtp.content_length/sizeof(uint32_t).
If there is no content, that is, the value of hbtp.content_length is 0, the
attributes of all channels are returned.**/
struct sdc_resolution {
uint32_t width;
uint32_t height;
};
#define SDC_PT_H264  96
#define SDC_PT_H265  265
#define SDC_PT_MJPEG 1002
struct sdc_venc_channel_ability {
uint32_t max_fps;           // Maximum frame rate supported by the channel.
uint32_t format[3]; // Encoding format supported by the channel. The value can
be 96 (H.264), 265 (H.265), or 1002 (MJPEG).
uint32_t resolution_num;  // Number of resolution types supported by the
channel.
struct sdc_resolution chn_resolution[0];  // All resolution types supported by
the channel.
};
struct sdc_venc_channel_info
{
struct sdc_venc_channel_param param;
uint32_t src_id;         // Data source. The data source uses the IP address
to identify the data source (local camera or other cameras) of the channel.
For the local multi-lens camera, the data source is identified by 127.0.0.x,
where x starts from 1 and is incremented by 1.
```

```
uint32_t subscriber_cnt; // Number of users who subscribe to data of this
channel. The value greater than 1 indicates that multiple users subscribe to
data of the same channel. Video data is transferred to users in zero-copy
mode. If a user modifies the data obtained from this channel, other users will
read the data after modification.
struct sdc_venc_channel_ability stchnability;  // Information about the
encoding capability supported by the channel.
};
/**When attributes of logical channels are queried in batches, the response
content is also returned in batches. The number of channels returned in
batches is calculated as follows: hbtp.content_length/sizeof(struct
sdc_venc_channel_info).**/
```

Reference example

```
#include "sdc.h"
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
int main(int argc,char* argv[])
{
int nret,i;
char buf[1024] = { 0 };
struct sdc_venc_channel_info* info;
struct sdc_common_head* head = (struct sdc_common_head*)buf;
/** query all channels' info */
head->version = SDC_VERSION; //0x5331
head->url = SDC_URL_VENC_CHANNEL; //0x02
head->method = SDC_METHOD_GET; //0x02
head->head_length = sizeof(*head);
int fd = open("/mnt/srvfs/video.iaas.sdc", O_RDWR);
if(fd < 0) goto fail;
nret = write(fd, head, head->head_length + head->content_length);
if(nret < 0) goto fail;
nret = read(fd,buf,sizeof(buf));
if(nret < 0 || head->code != SDC_CODE_200) goto fail;
/** deal with  all channels' info */
close(fd);
return 0;
fail:
exit(1); //fd will be closed after exit
}
```

I.2.6 VENC frame data subscription example

Content definition

```
uint32_t channel;
/**Multiple channels can be obtained. The number of channels is calculated as
follows: hbtp.content_length/sizeof(uint32_t).
After the subscription is successful, multiple packets are received
consecutively. The first response packet has no content.
The subsequent packets include VENC frame data. The structure is defined as
follows:**/
#define SDC_VENC_FRAME_I     0
#define SDC_VENC_FRAME_P    1
#define SDC_VENC_FRAME_B    2
struct sdc_venc_frame
{
uint64_t   addr_phy;
uint64_t   addr_virt; // Read-only.
uint64_t  size;
uint32_t    height;
uint32_t    width;
```

```
uint32_t   format;       //SDC_H264 OR SDC_H265
uint32_t   frame_type;   //SDC_VENC__FRAME_I/P/B
uint64_tcookie[8];    // Used for commissioning on the server.
}
struct sdc_venc_data
{
uint32_t channel;
uint32_t reserve;
uint64_t frame_pts;        // Frame timestamp.
uint64_t pts_sys;          // System timestamp.
struct sdc_venc_frame frame;
};
/**When channel data is subscribed in batches, the number of channels is
calculated as follows: hbtp.content_length/sizeof(struct sdc_venc_data).
Note: The trans_id field in the response headers of multiple packets is the
same as the trans_id field in the subscription request.**/
```

Reference Example

```
#include "sdc.h"
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
static void display_extend_head(struct sdc_extend_head* extend_head) {}
static void display_venc_data(struct sdc_venc_data* venc_data) {}
int main(int argc,char* argv[])
{
int nret,i,fd;
char buf[1024] = { 0 };
struct sdc_common_head* head = (struct sdc_common_head*) buf;
struct sdc_extend_head* extend_head;
struct sdc_venc_data* venc_data;
uint32_t* channel;
fd = open("/mnt/srvfs/video.iaas.sdc", O_RDWR);
if(fd < 0) goto fail;
head->version = SDC_VERSION;
head->url = SDC_URL_VENC_DATA;
head->method = SDC_METHOD_GET;
head->head_length = sizeof(*head);
/** The maximum number of cached VENC frames is 10. */
extend_head = (struct sdc_extend_head*)&buf[head->head_length];
extend_head->type = SDC_HEAD_VENC_CACHED_COUNT_MAX;
extend_head->length = sizeof(*extend_head);
extend_head->reserve = 10;
head->head_length += sdc_extend_head_length(extend_head);
/** Subscribe to data of channel 0. */
channel = (uint32_t*)&buf[head->head_length];
*channel = 0;
head->content_length = sizeof(*channel);
nret = write(fd, head, head->head_length + head->content_length);
if(nret < 0) goto fail;
for(;;) {
nret = read(fd, buf,sizeof(buf));
if(nret < 0) goto fail;
sdc_for_each_extend_head(head, extend_head) {
display_extend_head(extend_head);
}
for(i = 0, venc_data = (struct sdc_venc_data*)&buf[head->head_length]; i <
head->content_length / sizeof(*venc_data); ++i, ++venc_data) {
display_venc_data(venc_data);
}
/** free venc_data */
head->response = head->code = 0;
```

```
head->method = SDC_METHOD_DELETE;
(void)write(fd,head,head->head_length + head->content_length); // server
ignore extended headers
}
return 0;
fail:
exit(1);
}
```

I.3 Algorithm inference example

I.3.1 Model creation example

Content definition Example

```
If no extension header is defined, the MMZ mode is used.
#define NNIE_MODEL_CONTENT_MMZ 0
#define NNIE_MODEL_CONTENT_FILE 1
If the extension header SDC_HEAD_NNIE_MODEL_CONTENT_TYPE is set to 0 or not
defined, the content specifies the MMZ in the following format: struct
sdc_mmz.
If the extension header SDC_HEAD_NNIE_MODEL_CONTENT_TYPE is set to 1, the
content specifies the file name in the following format: char filename[].
hbtp.content_length indicates the length of the file name. Therefore, whether
the file name ends with NULL does not affect the function.
```

Reference Example

```
int SDC_LoadModel(unsigned int uiLoadMode, char *pucModelFileName,
SVP_NNIE_MODEL_S *pstModel)
{
int s32Ret = 0;
int ret = 0;
int u32TotalSize = 0;
struct sdc_extend_head* extend_head;
char buf[1024] = {0};
struct sdc_common_head *phead = (struct sdc_common_head *)buf;
unsigned int uFileSize;
struct sdc_mmz stMmzAddr;
if ((NULL == pstModel) || (NULL == pucModelFileName))
{
fprintf(stdout,"Err in SDC_LoadModel, pstModel or pucModelFileName is
null\n");
return -1;
}
fprintf(stdout,"Load model, pucModelFileName:%s!\n", pucModelFileName);
struct sdc_common_head head;
struct rsp_strcut {
struct sdc_common_head head;
SVP_NNIE_MODEL_S model;
}rsp_strcut_tmp;
struct iovec iov[2] = {
[0] = { .iov_base = buf, .iov_len = sizeof(struct sdc_common_head) +
sizeof(struct sdc_extend_head)},
[1] = { .iov_len  = MAX_MODULE_PATH}
};
//memset(&head, 0, sizeof(head));
phead->version        = SDC_VERSION;
phead->url            = SDC_URL_NNIE_MODEL;
phead->method         = SDC_METHOD_CREATE;
phead->head_length    = sizeof(struct sdc_common_head);
phead->content_length = MAX_MODULE_PATH;
```

```
/* If the mode is 0 and no extension header is carried, load the model from
the MMZ by default. */
if (uiLoadMode == 0)
{
FILE *fp = fopen(pucModelFileName, "rb");
if(fp == NULL)
{
fprintf(stdout,"modelfile fopen %s fail!\n", pucModelFileName);
return -1;
}
ret = fseek(fp,0L,SEEK_END);
if(ret != 0)
{
fprintf(stdout,"check nnie file SEEK_END, fseek fail.");
fclose(fp);
return -1;
}
uFileSize = ftell(fp);
ret = fseek(fp,0L,SEEK_SET);
if(0 != ret)
{
fprintf(stdout,"check nnie file SEEK_SET, fseek fail.");
fclose(fp);
return -1;
}
stMmzAddr.size = uFileSize;
ret = SDC_MmzAlloc(uFileSize, 0, &stMmzAddr); // param 2: 0 no cache, 1 cache
if(ret != stMmzAddr.size)
{
fprintf(stdout,"SDC_MmzAlloc ret %d, readsize %d", ret, stMmzAddr.size);
return -1;
}
ret = fread((HI_VOID*)(uintptr_t)stMmzAddr.addr_virt, 1, stMmzAddr.size, fp);
if(ret != stMmzAddr.size)
{
fprintf(stdout,"filesize %d, readsize %d", ret, stMmzAddr.size);
return -1;
}
/* Invoke the algorithm program to decode the input file. */
if(SDC_ModelDecript(&stMmzAddr))
{
fprintf(stdout,"SDC_ModelDecript Fail!",);
return -1;
}
iov[1].iov_base = &stMmzAddr;
iov[0].iov_len = sizeof(struct sdc_common_head);
}
else if (uiLoadMode == 1)/* If the mode is 1 and an extension header is
carried, specify the extension header parameter to load the model from the
MMZ. */
{
FILE *fp = fopen(pucModelFileName, "rb");
if(fp == NULL)
{
fprintf(stdout,"modelfile fopen %s fail!\n", pucModelFileName);
return -1;
}
ret = fseek(fp,0L,SEEK_END);
if(ret != 0)
{
fprintf(stdout,"check nnie file SEEK_END, fseek fail.");
fclose(fp);
return -1;
}
```

```
uFileSize = ftell(fp);
ret = fseek(fp,0L,SEEK_SET);
if(0 != ret)
{
fprintf(stdout,"check nnie file SEEK_SET, fseek fail.");
fclose(fp);
return -1;
}
stMmzAddr.size = uFileSize;
ret = SDC_MmzAlloc(uFileSize, 0, &stMmzAddr); // param 2: 0 no cache, 1 cache
if(ret != stMmzAddr.size)
{
fprintf(stdout,"SDC_MmzAlloc ret %d, readsize %d", ret, stMmzAddr.size);
return -1;
}
ret = fread((HI_VOID*)(uintptr_t)stMmzAddr.addr_virt, 1, stMmzAddr.size, fp);
if(ret != stMmzAddr.size)
{
fprintf(stdout,"filesize %d, readsize %d", ret, stMmzAddr.size);
return -1;
}
/* Invoke the algorithm program to decode the input file. */
if(SDC_ModelDecript(&stMmzAddr))
{
fprintf(stdout,"SDC_ModelDecript Fail!",);
return -1;
}
extend_head = (struct sdc_extend_head*)&buf[phead->head_length];
extend_head->type = 1;//NNIE_NNIE_MODEL_OP
extend_head->length = sizeof(*extend_head);
extend_head->reserve = 0;/* If this field is set to 0 or not carried, the
model is loaded from the memory. If this field is set to 1, the model is
loaded from a specified file. */
phead->head_length += sizeof(struct sdc_extend_head);
iov[1].iov_base = &stMmzAddr;
}
else /* If the mode is 2 and an extension header is carried, specify the
extension header parameter to load the model from a specified file. */
{
extend_head = (struct sdc_extend_head*)&buf[phead->head_length];
extend_head->type = 1;//NNIE_NNIE_MODEL_OP
extend_head->length = sizeof(*extend_head);
extend_head->reserve = 1;/* If this field is set to 0 or not carried, the
model is loaded from the memory. If this field is set to 1, the model is
loaded from a specified file. */
phead->head_length += sizeof(struct sdc_extend_head);
iov[1].iov_base = pucModelFileName;//pcModelName;
}
s32Ret = writev(fd_algorithm, iov, 2);
if (s32Ret < 0)
{
fprintf(stdout,"creat nnie,write to algorithm.iaas.sdc fail: %m\n");
}
/* After loading the model, release it immediately. */
if (uiLoadMode < 2)mmz_free(fd_config, &stMmzAddr);
s32Ret = read(fd_algorithm, &rsp_strcut_tmp, sizeof(rsp_strcut_tmp));
if(s32Ret == -1)
{
fprintf(stdout,"get_channel_data fail: %m\n");
return -1;
}
if(s32Ret > sizeof(rsp_strcut_tmp))
{
```

```
fprintf(stdout,"get_channel_data truncated, data len: %d > %zu\n", s32Ret,
sizeof(rsp_strcut_tmp));
return -1;
}
if (s32Ret < 0 || rsp_strcut_tmp.head.code != SDC_CODE_200 ||
rsp_strcut_tmp.head.content_length <= 0)
{
fprintf(stdout,"get nnie create response, read from algorithm.iaas.sdc
fail,s32Ret:%ld, code=%d,length=%d\n",
s32Ret, rsp_strcut_tmp.head.code, rsp_strcut_tmp.head.content_length);
}
else
{
s_stSsdModel.stModel = rsp_strcut_tmp.model;
memcpy(pstModel, &rsp_strcut_tmp.model,sizeof(SVP_NNIE_MODEL_S));
}
return s32Ret;
}
```

I.3.2 Model deletion example

```
int SDC_UnLoadModel(SVP_NNIE_MODEL_S *pstModel)
{
int nRet = -1;
if (NULL != pstModel)
{
struct sdc_common_head head;
struct iovec iov[2] = {
[0] = {.iov_base = &head , .iov_len = sizeof(head)},
[1] = {.iov_base = pstModel, .iov_len = sizeof(SVP_NNIE_MODEL_S)}
};
// fill head struct
memset(&head, 0, sizeof(head));
head.version    = SDC_VERSION;
head.url    = SDC_URL_NNIE_MODEL;
head.method      = SDC_METHOD_DELETE;
head.head_length    = sizeof(head);
head.content_length = sizeof(SVP_NNIE_MODEL_S);
nRet = writev(fd_algorithm, iov, sizeof(iov)/sizeof(iov[0]));
if (nRet < 0)
{
fprintf(stdout,"Errin SDC_UnLoadModel:failed to unload nnie module!\n");
}
}
else
{
fprintf(stdout,"Err in SDC_UnLoadModel:module pointer is NULL!\n");
}
return 0;
}
```

I.3.3 Model forward example

```
/**
* Compared with SVN_NNIE_FORWARD_CTRL_S, this interface has the following
advantages:
* 1. Users do not need to manage auxiliary memory segments, which simplifies
interface use.
* 2. Users do not need to specify NNIE_ID, which is scheduled by the server,
avoiding resource conflicts when multiple models are executed.
* 3. Multiple models share auxiliary memory segments to minimize memory
resource requirements.
*/
struct sdc_nnie_forward_ctrl
{
```

```
uint32_t netseg_id;
uint32_t max_batch_num;
uint32_t max_bbox_num;
uint32_t reserve;
};
struct {
SVP_NNIE_MODEL_S model;
struct sdc_nnie_forward_ctrl foward_ctl;
SVP_SRC_BLOB_S astSrc[16];
SVP_DST_BLOB_S astDst[16];
};

void SDC_Nnie_Forward(struct sdc_nnie_forward *p_sdc_nnie_forward)
{
int nRet;
struct sdc_common_head rsp_head;
struct sdc_common_head head;
struct iovec iov[2] = {
[0] = {.iov_base = &head, .iov_len = sizeof(head)},
[1] = {.iov_base = p_sdc_nnie_forward, .iov_len = sizeof(*p_sdc_nnie_forward)}
};
// fill head struct
memset(&head, 0, sizeof(head));
head.version    = SDC_VERSION;
head.url    = SDC_URL_NNIE_FORWARD;
head.method      = SDC_METHOD_GET;
head.head_length    = sizeof(head);
head.content_length = sizeof(*p_sdc_nnie_forward);
// write request
nRet = writev(fd_algorithm, iov, sizeof(iov)/sizeof(iov[0]));
if (nRet < 0)
{
fprintf(stdout,"Error:failed to write info to NNIE Forward!\n");
}
// read response
iov[0].iov_base = &rsp_head;
iov[0].iov_len = sizeof(rsp_head);
nRet = readv(fd_algorithm, iov, 1);
if (rsp_head.code != SDC_CODE_200 || nRet < 0)
{
fprintf(stdout,"Error:failed to read info from NNIE Forward!\n");
}
}
```

Bibliography

[b-ITU-T H.264] Recommendation ITU-T H.264 (2019), *Advanced video coding for generic audiovisual services*.

[b-ITU-T H.626] Recommendation ITU-T H.626 (2019), *Architectural requirements for video surveillance system*.

[b-ITU-T H.627] Recommendation ITU-T H.627 (2020), *Signalling and protocols for a video surveillance system*.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	**Non-telephone telecommunication services**
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling, and associated measurements and tests
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2021