NagraVision

Set-Top Box & Multimedia Unit

Conditional Access OpenTV

Application Programming Interface

V 1.8.0

✆ +41 (21) 732 03 11
📠 +41 (21) 732 03 00

Table Of Contents

List Of Figures

List Of Tables

1 Introduction

1.1 Purpose

This document provides the reader with the specification of the NagraVision Conditional Access Application Programming Interface for OpenTV. This API describes three blocks of functionalities: the alarms, the notifications and the functions.

1.2 Document History

Version	Date	Author	Description
1.8.0	20-Oct-2000	Bernard Krummenacher/ Patrick Schyrr	New template. Added new function `caGetMoreSystemInfo` giving the CAO version and smart card operator ID (`caGetSystemInfo` remains backward-compatible). Added new data types and data types manipulation functions.
1.7.0	Oct-2000	Marc Uldry	updated document revision for a new module release.
1.6.0	Nov-1999	Marc Uldry	Added specifications for the function `caGetSmartcardStatus` and revision after internal comments.
1.5.0	Aug-1998	Christian Bullat	Specified the audience of the document.
1.4.1	Jul-1998	Christian Bullat	Completed specifications for the function `caRegisterClient`.
1.4.0	Jun-1998	Christian Bullat	Revision after internal comments, improved comments, new template.
1.3.1	Apr-1998	Maxime Goeke	Revision after internal comments.
1.3.0	Mar-1998	Maxime Goeke	Revision after comments from customer.
1.2.0	Mar-1998	Maxime Goeke	Revision after comments from partner and decoder manufacturer.
1.1.0	Feb-1998	Maxime Goeke	Revision after comments from OpenTV and partner.
1.0.0	Jan-1998	Maxime Goeke	First issue

Table 1 - Document History

1.3 Definitions, Acronyms, and Abbreviations

Acronym	Definition

Abbreviation	
API	Application Programming Interface
ASCIIZ	Zero-terminated ASCII characters array
CA	Conditional Access
CAK	Conditional Access Kernel
DVB	Digital Video Broadcasting
EBNF	Extended Backus-Naur Form
EPG	Electronic Program Guide
EIT	Event Information Table
EMM	Entitlement Management Message
IPPV	Impulse Pay Per View
NASP	Nagra Advanced Security Processor
NIT	Network Information Table
PPV	Pay Per View
SDT	Service Description Table
SI	Service Information
STB	Set Top Box
UTC	Universal Time Co-ordinated

Table 2 - Definitions, Acronyms, and Abbreviations

1.4 Notational Conventions

- `Source code/ configuration file entries` are in Courier font.
- *Directories* and *filenames* are in italic type.

1.5 References

[1] NagraVision, “Digital Pay-TV System, System Overview”, V 1.1, February 1997.
[2] NagraVision, “Set Top Box, Conditional Access Task, Overview”, V 1.1, December 1999.
[3] NagraVision, “Set Top Box, Conditional Access Kernel, Application Programming Interface”, V2.3.0, October 2000.
[4] NagraVision, “DVB-SI Descriptors”, V 2.3.
[5] OpenTV, “Software Developer’s Kit 1.1, Getting started”, April 1997.
[6] OpenTV, “Software Developer’s Kit 1.1, Programmer’s guide”, April 1997.

1.6 Trademarks

Any company's or product name(s) found herein may be the trademarks or registered trademarks of their respective companies.

1.7 Accompanying files

Four files, providing code and definitions for this API, come with this document:

The first one, *calib.h*, describes the three blocks of functionalities of the CA, the alarms, the notifications and the functions, at the o-code level (OpenTV applicative level).

caotools.h and *caotools.c* provide functions that can be used to manipulate the cao data types at OpenTV applicative level and at native level; they are provided for developers' comfort and as such it is not mandatory to use them.

The last one, *calibx.h*, describes just one block of the functionalities of the CA, the alarms, at the native level (Control Task level). It includes *opentvx.h*, which shall be provided by the set-top box manufacturer.

1.8 Audience

This document is intended for both the OpenTV application developers and the STB Control Task developers.

1.8.1 OpenTV application developers

OpenTV applications developers will use the complete CAO API. They shall include the *calib.h* and optionally *caotools.h* files in their sources.

Pop-up applications developers will use the CA alarms, at the o-code level. They shall include the *calib.h* and optionally *caotools.h* files in their sources.

STB Control Task developers will use the CA alarms, at the native level. They shall include the *calibx.h* file in their sources.

Besides the processing of the alarm messages at the native level, the Control Task is very likely to launch pop-up applications to warn the user that something is going wrong with the CA.

2 Application programming interface

In this section appears the Conditional Access Application Programming Interface for OpenTV applications as specified today, based on the currently available features of the NagraVision CA system.

2.1 Jump table

The manufacturer has to create a header file named *calibnb.h*. This file, that will be included in the Nagravision header files *calib.h* and *calibx.h*, must define the constant `calib_number`. The value of this constant is defined by the operator. The following example shows the definition of the constant `calib_number` as it should appear in the *calibnb.h*. In this example, its value is set to 8:

```
#ifndef         CALIBNB_H
#define         CALIBNB_H
#define         calib_number            8
#endif
```

2.2 Symbolic constants

The symbolic constants shown here below are defined in the file *calib.h*. Their use is presented and explained throughout the following paragraphs.

Message class

`MSG_CLASS_CA`	Already defined by OpenTV in file *opentv.h* (o-code level).
`OPTV_CA_CLASS`	Already defined by OpenTV in file *opentvx.h* (native level).

Message types

```
MSG_TYPE_CA_ALARM
MSG_TYPE_CA_COMMAND
MSG_TYPE_CA_NOTIFICATION
```

Message fields

```
INFO_CA_ALARM
INFO_CA_COMMAND
INFO_CA_NOTIFICATION
INFO_CA_RESULT
INFO_CA_STATUS
```

`INFO_CA_ALARM` message field values

```
CA_SMART_CARD_BLACKLISTED
CA_SMART_CARD_COMMUNICATION
CA_SMART_CARD_GOOD
CA_SMART_CARD_MUTE
CA_SMART_CARD_NEVER_PAIRED
CA_SMART_CARD_NOT_INSERTED
CA_SMART_CARD_NOT_PAIRED
CA_SMART_CARD_NOT_RECOGNIZED
CA_SMART_CARD_SUSPENDED
```

`INFO_CA_COMMAND` message field values

```
CA_GET_CREDIT_INFO
CA_GET_DECODER_MESSAGE
CA_GET_EVENT_INFO
CA_GET_LOCATION_INFO
CA_GET_NUM_CREDITS
CA_GET_NUM_PURCHASED_PPV_EVENTS
CA_GET_PURCHASED_PPV_EVENT_INFO
CA_GET_SYSTEM_INFO
CA_PURCHASE_IPPV_EVENT
CA_REGISTER_CLIENT
CA_UNREGISTER_CLIENT
CA_GET_SMART_CARD_STATUS
CA_GET_MORE_SYSTEM_INFO
```

`INFO_CA_NOTIFICATION` message field values

```
CA_CREDIT_INFO_CHANGED
CA_CURRENT_ACCESS_CHANGED
CA_DECODER_MESSAGE_RECEIVED
CA_EVENT_INFO_CHANGED
CA_LOCATION_INFO_CHANGED
CA_PURCHASED_PPV_EVENT_INFO_CHANGED
CA_SYSTEM_INFO_CHANGED
```

`INFO_CA_STATUS` message field values

```
CA_BAD
CA_CREDIT_NOT_FOUND
CA_CREDIT_SUSPENDED
CA_GOOD
CA_NOT_ENOUGH_CREDIT
CA_NOT_ENOUGH_MEMORY
CA_NOT_IPPV_EVENT
```

Command return values

```
CA_COMMAND_ACCEPTED
CA_COMMAND_REJECTED
```

Accesses

```
CA_ACCESS_BLACKOUT
CA_ACCESS_CLEAR
CA_ACCESS_DENIED
CA_ACCESS_FREE
CA_ACCESS_GRANTED
```

Types

```
CA_TYPE_IPPV
CA_TYPE_PPV
CA_TYPE_SUB
CA_TYPE_CLEAR
```

Flags

```
CA_FLAG_SUSPENDED
CA_FLAG_WATCHED
```

Smartcard statuses

```
CA_SMART_CARD_BLACKLISTED
CA_SMART_CARD_COMMUNICATION
CA_SMART_CARD_GOOD
CA_SMART_CARD_MUTE
CA_SMART_CARD_NEVER_PAIRED
CA_SMART_CARD_NOT_INSERTED
CA_SMART_CARD_NOT_PAIRED
CA_SMART_CARD_NOT_RECOGNIZED
CA_SMART_CARD_SUSPENDED
```

Sizes

`CA_COMPONENT_VERSION_SIZE`	(Value=50)
`CA_DECODER_MESSAGE_SIZE`	(Value=50)
`CA_EVENT_NAME_SIZE`	(Value=17)
`CA_PROJECT_NAME_SIZE`	(Value =10)
`CA_SERIAL_NUMBER_SIZE`	(Value=15)
`CA_SERVICE_NAME_SIZE`	(Value=10)
`CA_SOFTWARE_VERSION_SIZE`	(Value=20)

2.3 Alarms

This functionality is intended for STB Control Task developers. They will use the CA alarms at the native level and then include the *calibx.h* file in their sources.

The alarm messages are not sent directly to the application but rather to the control task, to warn it that something is going wrong with regard to the smart card. It is then up to the control task to decide what to do with the received alarm messages. Very likely, the control task should launch pop-up applications and forward the alarm messages to them, so as to display banners on the screen inviting the user to cancel the error conditions. In this case, pop-up applications developers will use the CA alarms at the o-code level, and then include the *calib.h* file in their sources.

The software layer that implements this API is not involved in any way in the forwarding of the alarm messages to the pop-up applications. Still it provides these applications with all the necessary symbolic constants so as to correctly identify the alarm messages. Each alarm message is described in the next paragraphs.

2.3.1 Smart card blacklisted

When the smart card that is inserted in the decoder is detected as a blacklisted smart card, the control task receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_ALARM`. The message field `INFO_CA_ALARM` is set to `CA_SMART_CARD_BLACKLISTED`.

A blacklisted smart card is irremediably disabled. A smart card might typically be blacklisted for instance in case of irregular use of the smart card.

2.3.2 Smart card communication

When the smart card that is inserted in the decoder does not allow an error-free communication, the control task receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_ALARM`. The message field `INFO_CA_ALARM` is set to `CA_SMART_CARD_COMMUNICATION`.

2.3.3 Smart card good

When all error conditions are cancelled and everything is going fine again with regard to the smart card, the control task receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_ALARM`. The message field `INFO_CA_ALARM` is set to `CA_SMART_CARD_GOOD`.

2.3.4 Smart card mute

When the smart card that is inserted in the decoder does not allow any communication at all, the control task receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_ALARM`. The message field `INFO_CA_ALARM` is set to `CA_SMART_CARD_MUTE`.

This alarm message is typically generated when the smart card is inserted upside-down.

2.3.5 Smart card never paired

When the smart card that is inserted in the decoder has never been paired with any decoder, the control task receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_ALARM`. The message field `INFO_CA_ALARM` is set to `CA_SMART_CARD_NEVER_PAIRED`.

2.3.6 Smart card not inserted

When the smart card is extracted from the decoder, the control task receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_ALARM`. The message field `INFO_CA_ALARM` is set to `CA_SMART_CARD_NOT_INSERTED`.

2.3.7 Smart card not paired

When the smart card that is inserted in the decoder is not paired with that decoder but with another one, the control task receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_ALARM`. The message field `INFO_CA_ALARM` is set to `CA_SMART_CARD_NOT_PAIRED`.

2.3.8 Smart card not recognized

When the smart card that is inserted in the decoder is not recognized as a NagraVision smart card, the control task receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_ALARM`. The message field `INFO_CA_ALARM` is set to `CA_SMART_CARD_NOT_RECOGNIZED`.

2.3.9 Smart card suspended

When the smart card that is inserted in the decoder is detected as a suspended smart card, the control task receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_ALARM`. The message field `INFO_CA_ALARM` is set to `CA_SMART_CARD_SUSPENDED`.

Unlike a blacklisted smart card, a suspended smart card is not irremediably disabled. It might recover from suspension thanks to an appropriate EMM. A smart card might typically be suspended if the user is a bad payer.

2.3.10 Example of an alarm sequence

Assume that a NagraVision smart card is inserted in the decoder and that everything is going fine.

Suppose that the user performs the following actions: First, he extracts his smart card from the decoder; Second, he inserts his smart card upside-down into the decoder; Third he extracts his smart card from the decoder again; Fourth he finally inserts his smart card into the decoder correctly. Here are the successive alarm messages the control task will receive: First, `CA_SMART_CARD_NOT_INSERTED`; Second `CA_SMART_CARD_MUTE`; Third, `CA_SMART_CARD_NOT_INSERTED`; Fourth, `CA_SMART_CARD_GOOD`.

2.3.11 Structure of an alarm message

The description below is given as a guide. It represents the fields found in an alarm message structure:

For more details, please refer to [6] "§2.2.1.5.Icon Gadget Message Handler", p2-10, and to [6] "§2.5.Messages and Message Handlers", p2-37.

```
optv_message Message;

Message.class;
Message.type;
Message.INFO_CA_ALARM;
```

2.3.12 Treatment of an alarm message

The description below is given as a guide:

An alarm message is received and treated by the control task. See the instruction "`case MSG_TYPE_CA_ALARM:`"

```
optv_message Message;

for(;;){
  /*the Control Task is waiting for the next message:*/
  user_queue_wait_message(&Message);
  switch(Message.class){
  .
  .
  .
  case OPTV_CA_CLASS:{
    switch(Message.type){
    .
    .
    case MSG_TYPE_CA_ALARM:{
      switch(Message.INFO_CA_ALARM){
      case CA_SMART_CARD_BLACKLISTED:{
        processCaSmartCardBlacklisted();
        break;}
      case CA_SMART_CARD_COMMUNICATION:{
        processCaSmartCardCommunication();
        break;}
      .
      .
      .
      case CA_SMART_CARD_SUSPENDED:{
        processCaSmartCardSuspended();
        break;}
      }/*end switch*/
      break;}
    .
    .
```

```
      }/*end switch*/
      break;}
    .
    .
    .
    }/*end switch*/
  }/*end for*/
```

2.4 Notifications

This functionality is intended for OpenTV application developers. They will use the CA notifications at the o-code level and then include the *calib.h* file in their sources.

Notification messages are sent asynchronously to the application to warn it that something happened with regard to the NagraVision CA system. The sending of most of the notification messages is triggered by the effect of an EMM on some records of the NagraVision smart card. Nevertheless, notifications might be generated even if information did not actually change. In most cases, when the application receives a notification message, it should call some of the functions provided in this API to update its internal information according to the latest modifications of the data base in the NagraVision smart card.

2.4.1 Credit

When the NagraVision smart card makes a change in the credit records, the application can get notified. It then receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_NOTIFICATION`. The message field `INFO_CA_NOTIFICATION` is set to `CA_CREDIT_INFO_CHANGED`.

This notification message is generated when PPV events are purchased, when the user asks for more credit, or when any other operation that affects the credit records occurs (for example a new smart card insertion). The application should then use the functions `caGetCreditInfo` and `caGetNumCredits` to update its internal information.

2.4.2 Current access

There are two categories of accesses. The accesses `CA_ACCESS_CLEAR`, `CA_ACCESS_FREE` and `CA_ACCESS_GRANTED` make up the first category, the one that grants access. The accesses `CA_ACCESS_BLACKOUT` and `CA_ACCESS_DENIED` make up the second category, the one that denies access.

When the current access to the currently watched event changes from one category to the other, according to the access records of the NagraVision smart card, the application can get notified. It then receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_NOTIFICATION`. The message field `INFO_CA_NOTIFICATION` is set to `CA_CURRENT_ACCESS_CHANGED`.

For example, if the user changes from a channel where he has access to a channel where he does not have access, this notification message is generated. If the user goes back to the channel where he has access, then this notification message is generated again.

As another example, if the user is watching a service that goes through a transition from a free access event to a PPV event to which the user does not have access, this notification message is generated. In the particular case where this PPV event has a free access preview time at its beginning, this notification message is generated only at the end of the preview time. In the same situation, but if the user has paid to watch this PPV event, then this notification message is not generated, since a change from a free access to a granted access does not involve a change from an access category to the other.

2.4.3 Decoder message

When the NagraVision CA system receives an EMM that contains a message intended for the decoder, the application can get notified. It then receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_NOTIFICATION`. The message field `INFO_CA_NOTIFICATION` is set to `CA_DECODER_MESSAGE_RECEIVED`.

The NagraVision CA system does not impose any semantics for these messages, but rather only provides the application with the transport mechanism that allows it to receive secured, and possibly addressed, messages.

Decoder messages are internally queued into a first-in-first-out structure, so as to allow several messages to be received. Upon reception of this notification message, the application should call the function `caGetDecoderMessage` to retrieve the decoder message from the queue. As long as the queue is not empty, this notification message is generated again, after each successive call to `caGetDecoderMessage`. Whenever the queue gets full, the incoming messages are simply discarded. The default queue size is 10, but it can be adapted to the network's requirements.

2.4.4 Event

When the NagraVision smart card makes a change in the access records, the application can get notified. It then receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_NOTIFICATION`. The message field `INFO_CA_NOTIFICATION` is set to `CA_EVENT_INFO_CHANGED`.

This notification message is generated when PPV events or subscriptions are purchased, or when any other operation that affects the access records occurs (for example a new smart card insertion). The application should then use the function `caGetEventInfo` to update its internal information.

2.4.5 Location

When the NagraVision smart card makes a change in the ZIP code record or in the delta time record, or when any other operation that affects the ZIP code record or the delta time record occurs (for example a new smart card insertion) the application can get notified. It then receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_NOTIFICATION`. The message field `INFO_CA_NOTIFICATION` is set to `CA_LOCATION_INFO_CHANGED`.

The application should then use the function `caGetLocationInfo` to update its internal information.

2.4.6 Purchased PPV event

When the NagraVision smart card makes a change in the PPV records, the application can get notified. It then receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_NOTIFICATION`. The message field `INFO_CA_NOTIFICATION` is set to `CA_PURCHASED_PPV_EVENT_INFO_CHANGED`.

This notification message is generated when PPV events are purchased or collected for billing, or when any other operation that affects the PPV records occurs (for example a new smart card insertion). The application should then use the functions `caGetNumPurchasedPpvEvents` and `caGetPurchasedPpvEventInfo` to update its internal information.

2.4.7 System

When a NagraVision smart card is inserted into the decoder, the application can get notified. It then receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_NOTIFICATION`. The message field `INFO_CA_NOTIFICATION` is set to `CA_SYSTEM_INFO_CHANGED`.

This notification message is generated when the smart card that is inserted into the decoder is different from the one that was previously extracted from it. It is not generated if the smart card that is inserted into the decoder is the same as the one that was previously extracted from it.

The application should then use the function `caGetSystemInfo` to update its internal information.

In addition to the `CA_SYSTEM_INFO_CHANGED` notification message, the application might as well receive some other notification messages, like `CA_CREDIT_INFO_CHANGED` or `CA_EVENT_INFO_CHANGED` for instance. This is due to the fact that when a smart card replaces another one in the decoder, their data bases are very likely to be different as well.

2.4.8 Structure of a notification message

The description below is given as a guide. It represents the fields found in a notification message structure:

For more details, please refer to [6] "§2.2.1.5.Icon Gadget Message Handler", p2-10, and to [6] "§2.5.Messages and Message Handlers", p2-37.

```
o_message Message;

Message.msg_class;
Message.type;
Message.INFO_CA_NOTIFICATION;
```

2.4.9 Treatment of a notification message

The description below is given as a guide:

A notification message is received and treated by an OpenTV application. See the instruction "`case MSG_TYPE_CA_NOTIFICATION:`"

```
o_message Message;

for(;;){
  /*the OpenTV application is waiting for the next message:*/
  O_ui_get_event_wait(&Message);
  switch(O_msg_class(&Message)){
  case MSG_CLASS_CA:{
    switch(O_msg_type(&Message)){
   case MSG_TYPE_CA_NOTIFICATION:{
     switch(Message.INFO_CA_NOTIFICATION){
     case CA_CREDIT_INFO_CHANGED:{
       processCaCreditInfoChanged();
       break;}
     case CA_CURRENT_ACCESS_CHANGED:{
       processCaCurrentAccessChanged();
       break;}
     .
     .
     .
     case CA_SYSTEM_INFO_CHANGED:{
       processCaSystemInfoChanged();
       break;}
     }/*end switch*/
     break;}
   case MSG_TYPE_CA_COMMAND:{
     .
     .
     break;}
```

```
      }/*end switch*/
      break;}
    case :{
      .
      .
      break;}
    case :{
      .
      .
      break;}
    }/*end switch*/
  }/*end for*/
```

2.5 Functions

This functionality is intended for OpenTV application developers. They will use the CA functions at the o-code level and then include the *calib.h* file in their sources.

As already explained (please refer “§2.1 Jump table”), the *calib.h* file includes the *calibnb.h* that must be created by the manufacturer. This *calibnb*.h file must define the constant `calib_number`.

These functions are used either to get informations about the CA status or to react to the reception of a notification message, indicating a change in the CA status.

A function call has two consequences:

The first consequence is the immediate returned value that indicates whether the function call could successfully be initiated or not.

The second consequence is a command message, sent by the CA and received asynchronously by the application. This message will be sent, first if the initiation of the function call succeeds, and then after the complete execution of the function call. This command message contains, among others, the requested data and a boolean indicating the validity of this data.

A new call to the same function can only be performed after the reception of the message of the previous call to that function. In other words, the OpenTV application’s developer has to wait for the returned message after a function call, to call the same function again.

On the other hand, a burst call of different functions is possible.

Memory allocation is not performed by this API. Typically an OpenTV application calls a function of this API, passing a pointer as parameter. The memory shall be allocated by the calling application (example: see prototype of function `caGetDecoderMessage`).

2.5.1 Structure of a command message

The description below is given as a guide. It represents the fields found in a command message structure:

For more details, please refer to [6] “§2.2.1.5.Icon Gadget Message Handler”, p2-10, and to [6] “§2.5.Messages and Message Handlers”, p2-37.

```
o_message Message;

Message.msg_class;
Message.type;
Message.INFO_CA_COMMAND;
Message.INFO_CA_STATUS;
Message.INFO_CA_RESULT;
```

2.5.2 Treatment of a command message

The description below is given as a guide:

For more details, please refer to [5] “§1.4.3.Main Message Loop”, p1-6, and to [5] “§1.6.2.Shared Message Handler”, p1-10.

A command message is received and treated by an OpenTV application. See the instruction “`case MSG_TYPE_CA_COMMAND:`”

```
o_message Message;

for(;;){
  /*the OpenTV application is waiting for the next message:*/
  O_ui_get_event_wait(&Message);
  switch(O_msg_class(&Message)){
  case MSG_CLASS_CA:{
    switch(O_msg_type(&Message)){
    case MSG_TYPE_CA_NOTIFICATION:{
      .
      .
      break;}
    case MSG_TYPE_CA_COMMAND:{
      switch(Message.INFO_CA_COMMAND){
      case CA_GET_CREDIT_INFO:{
        switch(Message.INFO_CA_STATUS){
        case CA_GOOD:{
          processCreditInfo(Message.INFO_CA_RESULT);
          break;}
        case CA_BAD:{
          processCreditInfo(NULL);
          break;}
        }/*end switch*/
        break;}
      case CA_GET_DECODER_MESSAGE:{
        switch(Message.INFO_CA_STATUS){
        case CA_GOOD:{
          processDecoderMessage(Message.INFO_CA_RESULT);
          break;}
        case CA_BAD:{
          processDecoderMessage(NULL);
          break;}
        }/*end switch*/
        break;}
      .
      .
      case CA_UNREGISTER_CLIENT:{
        switch(Message.INFO_CA_STATUS){
        case CA_GOOD:{
          processUnregisterClient(CA_GOOD);
          break;}
        case CA_BAD:{
          processUnregisterClient(CA_BAD);
          break;}
        }/*end switch*/
        break;}
      }/*end switch*/
      break;}
    }/*end switch*/
    break;}
  case :{
    .
```

```
      .
      break;}
    }/*end switch*/
  }/*end for*/
```

2.5.3 caGetCreditInfo

Description

Gives information about the ith credit record.

Prototype

```
typedef struct
{
  unsigned long int providerId;
  unsigned long int credit;
  unsigned long int flags;
} TCaCreditInfo;

Int caGetCreditInfo
(
  unsigned long int xIndex,
  TCaCreditInfo* pxInfo
);
```

Arguments

`xIndex`	In: An index in the list of credit records.
`pxInfo`	In: A pointer where to store the information.

Return value

`CA_COMMAND_ACCEPTED`	If the command could successfully be initiated, that is to say, whether the status of the smart card is equivalent to `CA_SMART_CARD_GOOD` or to `CA_SMART_CARD_NEVER_PAIRED` or to `CA_SMART_CARD_NOT_PAIRED` or to `CA_SMART_CARD_SUSPENDED`.
`CA_COMMAND_REJECTED`	Otherwise, that is to say, whether the status of the smart card is equivalent to `CA_SMART_CARD_BLACKLISTED` or to `CA_SMART_CARD_COMMUNICATION` or to `CA_SMART_CARD_MUTE` or to `CA_SMART_CARD_NOT_INSERTED` or to `CA_SMART_CARD_NOT_-RECOGNIZED`.
`CA_COMMAND_REJECTED`	Also if the call of `caGetCreditInfo` is prior to the one of the function `caGetNumCredits`.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.
The message field `INFO_CA_COMMAND` is set to `CA_GET_CREDIT_INFO`. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the retrieval of the information about the credit record could complete successfully or not, respectively. The

message field `INFO_CA_STATUS` is also set to `CA_BAD` if the index `xIndex` is out of range. Upon successful completion of the command, the message field `INFO_CA_RESULT` is set to a copy of the pointer `pxInfo` to a `TCaCreditInfo` structure.
In this structure, the field `providerId` gives the ID of the service provider, the field `credit` gives the amount of credit available for that service provider, and the field `flags` gives some complementary information about the credit record.
The field `flags` is implemented as a bitmap. To test for the presence of a flag, use bit wise operators with the following possible values:

`CA_FLAG_SUSPENDED`	The credit record is suspended.

The range of the index `xIndex` is 0 to N-1, where N is the number of credit records given by the function `caGetNumCredits`, otherwise the message field `INFO_CA_STATUS` is set to `CA_BAD`, even if the call is accepted. This function has to be called at least once prior to any call to the `caGetCreditInfo` function.

See also

Notification `CA_CREDIT_INFO_CHANGED`
Function `caGetNumCredits`

2.5.4 caGetDecoderMessage

Description

Gives the next available decoder message.

Prototype

```
int caGetDecoderMessage
(
  unsigned char* pxMessage
);
```

Arguments

`pxMessage`	In: A pointer where to store the message.

Return value

`CA_COMMAND_ACCEPTED`	If the command could successfully be initiated, that is to say, whether the status of the smart card is equivalent to `CA_SMART_CARD_GOOD` or to `CA_SMART_CARD_NEVER_PAIRED` or to `CA_SMART_CARD_NOT_PAIRED` or to `CA_SMART_CARD_SUSPENDED`.
`CA_COMMAND_REJECTED`	Otherwise, that is to say, whether the status of the smart card is equivalent to `CA_SMART_CARD_BLACKLISTED` or to `CA_SMART_CARD_COMMUNICATION` or to `CA_SMART_CARD_MUTE` or to `CA_SMART_CARD_NOT_INSERTED` or to `CA_SMART_CARD_NOT_-RECOGNIZED`.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.
The message field `INFO_CA_COMMAND` is set to `CA_GET_DECODER_MESSAGE`. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the retrieval of the next available decoder message could complete successfully or not, respectively. The message field `INFO_CA_STATUS` is also set to `CA_BAD` if there is no message in queue. Upon successful completion of the command, the message field `INFO_CA_RESULT` is set to a copy of the pointer `pxMessage`. Decoder messages have a size of `CA_DECODER_MESSAGE_SIZE` bytes.

See also

Notification `CA_DECODER_MESSAGE_RECEIVED`

2.5.5 caGetEventInfo

Description

Gives information about an event.

Prototype

```
typedef struct
{
  unsigned long int access;
  unsigned long int type;
  unsigned long int flags;
  unsigned long int price;
  o_time_in previewTime;
} TCaEventInfo;

int caGetEventInfo
(
  const o_time* pxStartTime,
  const unsigned char* pxCaDesc,
  const unsigned char* pxPpvDesc,
  const unsigned char* pxBlackoutDesc,
  TCaEventInfo* pxInfo
);
```

Arguments

`pxStartTime`	In: The UTC start time of the event.
`pxCaDesc`	In: A pointer to the NagraVision private CA descriptor of the event.
`pxPpvDesc`	In: A pointer to the NagraVision private PPV descriptor of the event.
`pxBlackoutDesc`	In: A pointer to the NagraVision private blackout descriptor of the event.
`pxInfo`	In: A pointer where to store the information.

Return value

`CA_COMMAND_ACCEPTED`	If the command could successfully be initiated, that is to say, whether the status of the smart card is equivalent to `CA_SMART_CARD_GOOD` or to `CA_SMART_CARD_NEVER_PAIRED` or to `CA_SMART_CARD_NOT_PAIRED` or to `CA_SMART_CARD_-SUSPENDED`.
`CA_COMMAND_REJECTED`	Otherwise, that is to say, whether the status of the smart card is equivalent to `CA_SMART_CARD_BLACKLISTED` or to `CA_SMART_CARD_COMMUNICATION` or to `CA_SMART_CARD_MUTE` or to `CA_SMART_CARD_NOT_INSERTED` or to `CA_SMART_CARD_NOT_-RECOGNIZED`.

It is worth noting that the call of this function will always be accepted on a clear access channel, whatever the state of the smart card. In fact, in such an access type channel, the status of the smart card will always be considered as `CA_SMART_CARD_GOOD`.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.
The message field `INFO_CA_COMMAND` is set to `CA_GET_EVENT_INFO`. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the retrieval of the information about the event could complete successfully or not, respectively. Upon successful completion of the command, the message field `INFO_CA_RESULT` is set to a copy of the pointer `pxInfo` to a `TCaEventInfo` structure.
In this structure, the field `access` gives the access condition to the event, the field `type` gives the type of the event, the field `flags` gives some complementary information about the event, the field `price` gives the price of the event and the field `previewTime` gives the preview time of the event.
The possible values for the field `access` are:

`CA_ACCESS_BLACKOUT`	The smart card is in a blackout area.
`CA_ACCESS_CLEAR`	The event is not scrambled.
`CA_ACCESS_DENIED`	The access to the event is denied.
`CA_ACCESS_FREE`	The access to the event is free.
`CA_ACCESS_GRANTED`	The access to the event is granted.

The possible values for the field `type` are:

`CA_TYPE_IPPV`	The event is an IPPV event.
`CA_TYPE_PPV`	The event is a PPV event.
`CA_TYPE_SUB`	The event is part of a subscription.
`CA_TYPE_CLEAR`	The event is a clear event.

The field `flags` is implemented as a bitmap. To test for the presence of a flag, use bitwise operators with the following possible values:

`CA_FLAG_SUSPENDED`	The access to the event is suspended.

If the event is an IPPV or a PPV event, the fields `price` and `previewTime` are relevant. If the event is part of a subscription, the fields `price` and `previewTime` are irrelevant.
The field `pxStartTime` is mandatory; otherwise the call will be rejected (`CA_COMMAND_REJECTED`).

Private descriptors

The function `caGetEventInfo` expects three private descriptors: the CA descriptor, the PPV descriptor and the blackout descriptor. They give the characteristics of an event and are retrieved from the SDT and EIT tables.
If any of these descriptors is present in these tables, it must be passed to the function `caGetEventInfo`.
If the function `caGetEventInfo` is called without any descriptor, this means that no descriptor was available in the SDT or in the EIT tables.
The descriptors PPV descriptor and blackout descriptor are optional. None, or only one, or both can be present in the call to the function `caGetEventInfo`. If one descriptor is present, it must come with the CA descriptor in the call to the function `caGetEventInfo`. If both are present, they must come with the CA descriptor in the call to the function `caGetEventInfo`.
If the CA descriptor is missing, then the returned field `access` is set to `CA_ACCESS_CLEAR` even if the call is accepted (`CA_COMMAND_ACCEPTED`).
If a NagraVision private descriptor is not available for the event, then the corresponding pointer shall be set to `NULL`.

See also

Notification `CA_EVENT_INFO_CHANGED`
Notification `CA_PURCHASED_PPV_EVENT_INFO_CHANGED`
Function `caGetPurchasedPpvEventInfo`

2.5.6 caGetLocationInfo

Description

Gives information about the location of the decoder.

Prototype

```
typedef struct
{
  unsigned long int zipCode;
  o_time_in deltaTime;
} TCaLocationInfo;

int caGetLocationInfo
(
  TCaLocationInfo* pxInfo
);
```

Arguments

`pxInfo`	In: A pointer where to store the information.

Return value

`CA_COMMAND_ACCEPTED`	If the command could successfully be initiated, that is to say, whether the status of the smart card is equivalent to `CA_SMART_CARD_GOOD` or to `CA_SMART_CARD_NEVER_PAIRED` or to `CA_SMART_CARD_NOT_PAIRED` or to `CA_SMART_CARD_SUSPENDED`.
`CA_COMMAND_REJECTED`	Otherwise, that is to say, whether the status of the smart card is equivalent to `CA_SMART_CARD_BLACKLISTED` or to `CA_SMART_CARD_COMMUNICATION` or to `CA_SMART_CARD_MUTE` or to `CA_SMART_CARD_NOT_INSERTED` or to `CA_SMART_CARD_NOT_-RECOGNIZED`.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.

The message field `INFO_CA_COMMAND` is set to `CA_GET_LOCATION_INFO`. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the retrieval of the information about the location of the decoder could complete successfully or not, respectively. Upon successful completion of the command, the message field `INFO_CA_RESULT` is set to a copy of the pointer `pxInfo` to a `TCaLocationInfo` structure.
In this structure, the field `zipCode` gives the ZIP code, and the field `deltaTime` gives the delta time between the local time and the GMT time.

See also

Notification `CA_LOCATION_INFO_CHANGED`
Notification `CA_SYSTEM_INFO_CHANGED`
Function `caGetSystemInfo`

2.5.7 caGetNumCredits

Description

Gives the number of credit records.

Prototype

```
int caGetNumCredits
(
  unsigned long int* pxNum
);
```

Arguments

`pxNum`	In: A pointer where to store the number.

Return value

`CA_COMMAND_ACCEPTED`	If the command could successfully be initiated, that is to say, whether the status of the smart card is equivalent to `CA_SMART_CARD_GOOD` or to `CA_SMART_CARD_NEVER_PAIRED` or to `CA_SMART_CARD_NOT_PAIRED` or to `CA_SMART_CARD_SUSPENDED`.
`CA_COMMAND_REJECTED`	Otherwise, that is to say, whether the status of the smart card is equivalent to `CA_SMART_CARD_BLACKLISTED` or to `CA_SMART_CARD_COMMUNICATION` or to `CA_SMART_CARD_MUTE` or to `CA_SMART_CARD_NOT_INSERTED` or to `CA_SMART_CARD_NOT_-RECOGNIZED`.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.
The message field `INFO_CA_COMMAND` is set to `CA_GET_NUM_CREDITS`. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the retrieval of the number of credit records could complete successfully or not, respectively. Upon successful completion of the command, the message field `INFO_CA_RESULT` is set to a copy of the pointer `pxNum`.
This function has to be called at least once prior to any call to the function `caGetCreditInfo`.

See also

Notification `CA_CREDIT_INFO_CHANGED`
Function `caGetCreditInfo`

2.5.8 caGetNumPurchasedPpvEvents

Description

Gives the number of purchased PPV events.

Prototype

```
int caGetNumPurchasedPpvEvents
(
  unsigned long int* pxNum
);
```

Arguments

`pxNum`	In: A pointer where to store the number.

Return value

`CA_COMMAND_ACCEPTED`	If the command could successfully be initiated, that is to say, whether the status of the smart card is equivalent to `CA_SMART_CARD_GOOD` or to `CA_SMART_CARD_NEVER_PAIRED` or to `CA_SMART_CARD_NOT_PAIRED` or to `CA_SMART_CARD_SUSPENDED`.
`CA_COMMAND_REJECTED`	Otherwise, that is to say, whether the status of the smart card is equivalent to `CA_SMART_CARD_BLACKLISTED` or to `CA_SMART_CARD_COMMUNICATION` or to `CA_SMART_CARD_MUTE` or to `CA_SMART_CARD_NOT_INSERTED` or to `CA_SMART_CARD_NOT_RECOGNIZED`.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.
The message field `INFO_CA_COMMAND` is set to CA_GET_NUM_PURCHASED_PPV_EVENTS. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the retrieval of the number of purchased PPV events could complete successfully or not, respectively. Upon successful completion of the command, the message field `INFO_CA_RESULT` is set to a copy of the pointer `pxNum`.
This function has to be called at least once prior to any call to the function `caGetPurchasedPpvEventInfo`.

See also

Notification `CA_PURCHASED_PPV_EVENT_INFO_CHANGED`
Function `caGetPurchasedPpvEventInfo`
Function `caPurchaseIppvEvent`

2.5.9 caGetPurchasedPpvEventInfo

Description

Gives information about the Ith purchased PPV event.

Prototype

```
typedef struct
{
  o_time startTime;
  char serviceName[CA_SERVICE_NAME_SIZE+1];
  char eventName[CA_EVENT_NAME_SIZE+1];
  unsigned long int providerId;
  unsigned long int type;
  unsigned long int flags;
  unsigned long int price;
} TCaPurchasedPpvEventInfo;

int caGetPurchasedPpvEventInfo
(
  unsigned long int xIndex,
  TCaPurchasedPpvEventInfo* pxInfo
);
```

Arguments

`xIndex`	In: An index in the list of purchased PPV events.
`pxInfo`	In: A pointer where to store the information.

Return value

`CA_COMMAND_ACCEPTED`	If the command could successfully be initiated, that is to say, whether the status of the smart card is equivalent to `CA_SMART_CARD_GOOD` or to `CA_SMART_CARD_NEVER_PAIRED` or to `CA_SMART_CARD_NOT_PAIRED` or to `CA_SMART_CARD_SUSPENDED`.
`CA_COMMAND_REJECTED`	Otherwise, that is to say, whether the status of the smart card is equivalent to `CA_SMART_CARD_BLACKLISTED` or to `CA_SMART_CARD_COMMUNICATION` or to `CA_SMART_CARD_MUTE` or to `CA_SMART_CARD_NOT_INSERTED` or to `CA_SMART_CARD_NOT_-RECOGNIZED`.
`CA_COMMAND_REJECTED`	Also if the call of `caPurchaseIppvEvent` is prior to the one of the function `caGetNumPurchasedPpvEvents`.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.

The message field `INFO_CA_COMMAND` is set to `CA_GET_PURCHASED_PPV_EVENT_INFO`. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the retrieval of the information about the purchased PPV event could complete successfully or not, respectively. The message field `INFO_CA_STATUS` is also set to `CA_BAD` if the index `xIndex` is out of range. Upon successful completion of the command, the message field `INFO_CA_RESULT` is set to a copy of the pointer `pxInfo` to a `TCaPurchasedPpvEventInfo` structure.
In this structure, the field `startTime` gives the event start time, the fields `serviceName` and `eventName` give the service name and the event name, as ASCIIZ strings, respectively, the field `provider_ID` gives the ID of the service provider of the event, the field `type` gives the type of the event, the field `flags` gives some complementary information about the event, and the field `price` gives the price of the event.

The possible values for the field `type` are:

`CA_TYPE_IPPV`	The event is an IPPV event.
`CA_TYPE_PPV`	The event is a PPV event.

The field `flags` is implemented as a bitmap. To test for the presence of a flag, use bitwise operators with the following possible values:

`CA_FLAG_SUSPENDED`	The access to the event is suspended.
`CA_FLAG_WATCHED`	The event is watched.

The range of the index `xIndex` is 0 to N-1, where N is the number of purchased PPV events given by the function `CaGetNumPurchasedPpvEvents`. As already written, if this index is out of range, the message field `INFO_CA_STATUS` will be set to `CA_BAD`. This function has to be called at least once prior to any call to the `caGetPurchasedPpvEventInfo` function.

See also

Notification `CA_PURCHASED_PPV_EVENT_INFO_CHANGED`
Function `caGetNumPurchasedPpvEvents`
Function `caPurchaseIppvEvent`

2.5.10 caGetSystemInfo

Description

Gives information about the NagraVision CA system.

Prototype

```
typedef unsigned char TCaDecoderSwVersion[CA_SOFTWARE_VERSION_SIZE +1];

typedef struct
{
  unsigned long int systemId;
  char                    decoderSerialNumber[CA_SERIAL_NUMBER_SIZE +1];
  TCaDecoderSwVersion     decoderSoftwareVersion;
  char                    smartCardSerialNumber[CA_SERIAL_NUMBER_SIZE +1];
  char                    smartCardSoftwareVersion[CA_SOFTWARE_VERSION_SIZE +1];
} TCaSystemInfo;

int caGetSystemInfo
(
  TCaSystemInfo* pxInfo
);
```

Arguments

`pxInfo`	In: A pointer where to store the information.

Return value

`CA_COMMAND_ACCEPTED`	If the command could successfully be initiated.
`CA_COMMAND_REJECTED`	If the command could not be issued (insufficient resources, invalid parameter, function busy…).

Side effects

None

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated.

If so, the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_-COMMAND` upon completion of the command.

The message field `INFO_CA_COMMAND` is set to `CA_GET_SYSTEM_INFO`.

The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the retrieval of the information about the NagraVision CA system could complete successfully or not, respectively.
Upon successful completion of the command, the message field `INFO_CA_RESULT` points to the `TCaSystemInfo` structure given by the parameter `pxInfo`.

In this structure, the field `systemId` gives the system ID of the NagraVision CA system.

The `decoderSerialNumber` and `decoderSoftwareVersion` fields give the NagraVision serial number of the decoder and the version of the NagraVision software embedded in the decoder, respectively.

The `smartCardSerialNumber` and `smartCardSoftwareVersion` fields give the serial number and the version of the software of the NagraVision smart card, respectively. In case of smart card error, these fields will contain an empty string.

The `decoderSerialNumber` and `smartCardSerialNumber` fields are preformatted strings, ready to be displayed, containing the decimal coded serial numbers. The `decoderSoftwareVersion` and `smartCardSoftwareVersion` fields are preformatted strings, ready to be displayed.

The `decoderSoftwareVersion` contains the name of Nagravision CA project name and version, separated by a space character (for example “NICEBOX 1.3.12”). In older CAO versions, it was made of the CAK version and the project version, separated by a space character (for example “1.1.0 1.3.12”).

`TCaDecoderSwVersion` can be printed as an ASCIIZ string.

See also

Notification `CA_LOCATION_INFO_CHANGED`
Notification `CA_SYSTEM_INFO_CHANGED`
Function `caGetLocationInfo`
Function `caGetMoreSystemInfo`
Function `caDecoderSwVersionGetPrjName`
Function `caDecoderSwVersionGetPrjVersion`

2.5.11 caGetMoreSystemInfo

Description

Gives additional information about the NagraVision CA system. This function is provided to complement `caGetSystemInfo` while maintaining backward-compatibility.
It is designed to be future-proof, by allowing addition of queries while maintaining the compatibility.

Prototype

```
typedef enum
{
  CA_GET_CAO_VERSION,
  CA_GET_SMART_CARD_OPERATOR_ID
} TCaMoreSystemInfoSpecifier;

typedef unsigned char TCaComponentVersion[CA_COMPONENT_VERSION_SIZE +1];

typedef union
{
  TCaComponentVersion   caoVersion;
  unsigned long int     smartcardOperatorId;
} TCaMoreSystemInfoData;

typedef struct
{
  TCaMoreSystemInfoSpecifier  infoSpecifier;
  TCaMoreSystemInfoData       data;
} TCaMoreSystemInfoResponse;


int caGetMoreSystemInfo
(
  TCaMoreSystemInfoSpecifier   xInfoSpecifier,
  TCaMoreSystemInfoResponse*   pxInfo
);
```

Arguments

`xInfoSpecifier`	In: The kind of requested system information.
`pxInfo`	In: A pointer to a user-allocated `TCaMoreSystemInfoResp` structure to receive the requested information.

Return value

`CA_COMMAND_ACCEPTED`	If the command could successfully be initiated.
`CA_COMMAND_REJECTED`	If the command could not be issued (insufficient resources, invalid parameter, unknown specifier, function busy…).

Side effects

None

Comments

Several information types can be requested through this command; this is specified by the parameter `xInfoSpecifier`:

`CA_GET_CAO_VERSION`	Requests the version of this component (CAO); it can be used to determine what functions are currently available. The version can be obtained from the `caoVersion` field of the response.
`CA_GET_SMART_CARD_OPERATOR_ID`	Requests the ID of the operator responsible for this smart card; the version can be obtained from the `smartcardOperatorId` field of the response.

Elements of the type `TCaMoreSystemInfoSpecifier` will never be removed to ensure backward-compatibility.
`caGetMoreSystemInfo` is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated.
If so, the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_-COMMAND` upon completion of the command.

The message field `INFO_CA_COMMAND` is set to `CA_GET_MORE_SYSTEM_INFO`.

The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the retrieval of the additional information about the system could complete successfully or not, respectively.

Upon successful completion of the command, the message field `INFO_CA_RESULT` points to the `TCaMoreSystemInfoResponse` structure given by the parameter `pxInfo`. The attribute `infoSpecifier` of the response will be set to `xInfoSpecifier`.

`TCaComponentVersion` can be printed as a string. Its format complies with the following rule (EBNF syntax):

```
TCaComponentVersion  ::= digit {digit} '.' digit {digit} '.' digit {digit} '\0'
digit ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

For example: “1.3.25”

See also

Function `caGetSystemInfo`
Function `caComponentVersionIsLowerThan`

2.5.12 caPurchaseIppvEvent

Description

Purchases an IPPV event.

Prototype

```
int caPurchaseIppvEvent
(
  const o_time* pxStartTime,
  const char* pxServiceName,
  const char* pxEventName,
  const unsigned char* pxPpvDesc,
  const unsigned char* pxIemmDesc,
  unsigned long int* pxProviderId
);
```

Arguments

`pxStartTime`	In: The start time of the event.
`pxServiceName`	In: An ASCIIZ string that gives the name of the service.
`pxEventName`	In: An ASCIIZ string that gives the name of the event.
`pxPpvDesc`	In: A pointer to the NagraVision private PPV descriptor of the event.
`pxIemmDesc`	In: A pointer to the NagraVision private IEMM descriptor of the event.
`pxProviderId`	In: A pointer where to store the ID of the service provider of the event.

Return value

`CA_COMMAND_ACCEPTED`	If the command could successfully be initiated, that is to say, whether the status of the smart card is equivalent to `CA_SMART_CARD_GOOD` or to `CA_SMART_CARD_NEVER_PAIRED` or to `CA_SMART_CARD_NOT_PAIRED` or to `CA_SMART_CARD_SUSPENDED`.
`CA_COMMAND_REJECTED`	Otherwise, that is to say, whether the status of the smart card is equivalent to `CA_SMART_CARD_BLACKLISTED` or to `CA_SMART_CARD_COMMUNICATION` or to `CA_SMART_CARD_MUTE` or to `CA_SMART_CARD_NOT_INSERTED` or to `CA_SMART_CARD_NOT_-RECOGNIZED`.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.
The message field `INFO_CA_COMMAND` is set to `CA_PURCHASE_IPPV_EVENT`. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the purchase of the IPPV event could complete successfully or not, respectively. In the particular case where no credit record can be found in the NagraVision smart card, the message field `INFO_CA_STATUS` is set to `CA_CREDIT_NOT_FOUND`. In the particular case where the credit record for the service provider of the event is suspended, the message field `INFO_CA_STATUS` is set to `CA_CREDIT_SUSPENDED`. In the particular case where the amount of credit available for the service provider of the event is too low, the message field `INFO_CA_STATUS` is set to `CA_NOT_ENOUGH_CREDIT`. In the particular case where the memory in the NagraVision smart card is full, the message field `INFO_CA_STATUS` is set to `CA_NOT_ENOUGH_MEMORY`. Finally, in the particular case where the event is not an IPPV event, the message field `INFO_CA_STATUS` is set to `CA_NOT_IPPV_EVENT`.
If the command is accepted and the completion status of the command is different than `CA_CREDIT_NOT_FOUND`, the message field `INFO_CA_RESULT` is set to a copy of the pointer `pxProviderId` where the ID of the service provider of the event is stored. In the case where there is not enough credit to buy the event, this information might be used for instance to filter the credit records so as to find the incriminated credit.
If the command is accepted and the completion status of the command is equal to `CA_CREDIT_NOT_FOUND`, the message field `INFO_CA_RESULT` is irrelevant.

Private descriptors

The function `caPurchaseIppvEvent` expects two private descriptors: the PPV descriptor and the IEMM descriptor. They give the characteristics of an IPPV event and are retrieved from the EIT table.
All the 6 parameters are mandatory.
If the function `caPurchaseIppvEvent` is called and if one of the descriptors is missing, then the returned result is set to `CA_COMMAND_REJECTED`.

See also

Notification `CA_EVENT_INFO_CHANGED`
Notification `CA_PURCHASED_PPV_EVENT_INFO_CHANGED`
Function `caGetEventInfo`
Function `caGetNumPurchasedPpvEvents`
Function `caGetPurchasedPpvEventInfo`

2.5.13 caGetSmartcardStatus

Description

Gives the status of the smart card.

Prototype

```
typedef struct
{
  unsigned long int status;
} TCaSmartcardStatus;
int caGetSmartcardStatus
(
  TCaSmartcardStatus* pxStatus
);
```

Arguments

`pxStatus`	In: A pointer where to store the information.

Return value

`CA_COMMAND_ACCEPTED`	If the command could successfully be initiated,
`CA_COMMAND_REJECTED`	otherwise.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.
The message field `INFO_CA_COMMAND` is set to CA_GET_SMARTCARD_STATUS. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the retrieval of the information about the status of the smart card could complete successfully or not, respectively. Upon successful completion of the command, the message field `INFO_CA_RESULT` is set to a copy of the pointer `pxStatus` to a `TCaSmartcardStatus` structure.
The field `pxStatus` corresponds to one of the following values:

```
CA_SMART_CARD_BLACKLISTED
CA_SMART_CARD_COMMUNICATION
CA_SMART_CARD_GOOD
CA_SMART_CARD_MUTE
CA_SMART_CARD_NEVER_PAIRED
CA_SMART_CARD_NOT_INSERTED
CA_SMART_CARD_NOT_PAIRED
CA_SMART_CARD_NOT_RECOGNIZED
CA_SMART_CARD_SUSPENDED
```

See also

`INFO_CA_ALARM`	message field values

2.5.14 caRegisterClient

Description

Allows an application to indicate that it wants to receive the NagraVision CA system notification messages.

Prototype

```
int caRegisterClient
(
  void
);
```

Arguments

None.

Return value

`CA_COMMAND_ACCEPTED`	If the command could successfully be initiated
`CA_COMMAND_REJECTED`	otherwise.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so, the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.
The message field `INFO_CA_COMMAND` is set to `CA_REGISTER_CLIENT`. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the registering of the client could complete successfully or not, respectively. Upon successful completion of the command, the NagraVision CA system notification messages are routed to the application.
This function is idempotent. It means that, even if this function is called several times in a row, the effect will not change. In other words, the actual registration occurs one time and can't be multiple.

See also

Function `caUnregisterClient`

2.5.15 caUnregisterClient

Description

Allows an application to indicate that it does not want to receive the NagraVision CA system notification messages any more.

Prototype

```
int caUnregisterClient
(
  void
);
```

Arguments

None.

Return value

`CA_COMMAND_ACCEPTED`	If the command could successfully be initiated.
`CA_COMMAND_REJECTED`	otherwise.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. The application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.
The message field `INFO_CA_COMMAND` is set to `CA_UNREGISTER_CLIENT`. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the registering of the client could complete successfully or not, respectively. Upon successful completion of the command, the NagraVision CA system notification messages are not routed to the application any more.
This function is idempotent. It means that, even if this function is called several times in a row, the effect will not change. In other words, the actual un-registration occurs one time and can't be multiple.

See also

Function `caRegisterClient`

2.6 Data type manipulation functions

This chapter gives information about data type manipulation functions to be used with the CAO. These functions are defined in *caotools.h*; they are implemented in *caotools.c*.

2.6.1 caDecoderSwVersionGetPrjVersion

Description

Extracts the Nagravision CA project version from the decoder software version.

Prototype

```
int caDecoderSwVersionGetPrjVersion
(
  const TCaDecoderSwVersion   xDecoderSwVersion,
        TCaComponentVersion   xProjectVersion
);
```

Arguments

`xDecoderSwVersion`	in: decoder software version to be parsed
`xProjectVersion`	out: resulting project version

Return value

`CA_COMMAND_ACCEPTED`	If the function succeeded.
`CA_COMMAND_REJECTED`	otherwise.

Side effects

None.

Comments

This function is synchronous.

See also

Function `caDecoderSwVersionGetPrjName`
Function `caGetSystemInfo`
Function `caComponentVersionIsLowerThan`

2.6.2 caDecoderSwVersionGetPrjName

Description

Extracts the Nagravision CA project name from the decoder software version.

Prototype

```
typedef unsigned char TCaProjectName[CA_PROJECT_NAME_SIZE +1];

int caDecoderSwVersionGetPrjName
(
  const TCaDecoderSwVersion   xDecoderSwVersion,
        TCaProjectName        xProjectName
);
```

Arguments

`xDecoderSwVersion`	in: decoder software version to be parsed
`xProjectVersion`	out: resulting project name

Return value

`CA_COMMAND_ACCEPTED`	If the function succeeded
`CA_COMMAND_REJECTED`	otherwise

Side effects

None.

Comments

This function is synchronous.
`TCaProjectName` can be printed as an ASCIIZ string.

See also

Function `caGetSystemInfo`

Function `caDecoderSwVersionGetPrjVersion`

2.6.3 caComponentVersionIsLower

Description

Compares two component versions.

Prototype

```
int caComponentVersionIsLowerThan
(
  const TCaComponentVersion   xVersion1,
  const TCaComponentVersion   xVersion2,
        bool*                 pxIsLower
);
```

Arguments

`xVersion1`	in: first version to be compared
`xVersion2`	in: second version to be compared
`pxIsLower`	out: `xVersion1 < xVersion2`

Return value

`CA_COMMAND_ACCEPTED`	If input parameters are correct
`CA_COMMAND_REJECTED`	otherwise

Side effects

None.

Comments

This function is synchronous.

See also

Function `caGetMoreSystemInfo`

Function `caDecoderSwVersionGetPrjVersion`

3 API’s future extensions

In this section appear the future extensions to the CA API for OpenTV applications, based on the planned new features of the NagraVision CA system. Be aware of the fact that every information given in this section is subject to modification, without any notice.

3.1 Symbolic constants

The symbolic constants shown here below are planned to be defined for use in future extensions. Their use is presented and explained throughout the following paragraphs.

`INFO_CA_COMMAND` message field values

```
CA_CLOSE_RETURN_CHANNEL
CA_DECRYPT_BUFFER
CA_ENCRYPT_BUFFER
CA_GET_CURRENCY_INFO
CA_OPEN_RETURN_CHANNEL
CA_READ_RETURN_CHANNEL
CA_WRITE_RETURN_CHANNEL
```

Sizes

`CA_SYMBOL_SIZE` (Value=5)

3.2 Functions

3.2.1 caCloseReturnChannel

Description

Closes a connection on the return channel.

Prototype

```
int caCloseReturnChannel
(
  unsigned long int xChannelId
);
```

Arguments

`xChannelId`	In: The ID of the channel.

Return value

`CA_COMMAND_ACCEPTED`	If the command could successfully be initiated,
`CA_COMMAND_REJECTED`	otherwise.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.
The message field `INFO_CA_COMMAND` is set to `CA_CLOSE_RETURN_CHANNEL`. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the connection on the return channel could be closed successfully or not, respectively.

See also

Function `caOpenReturnChannel`
Function `caReadReturnChannel`
Function `caWriteReturnChannel`

3.2.2 caDecryptBuffer

Description

Decrypts a chunk of data.

Prototype

```
int caDecryptBuffer
(
  unsigned long int xProviderId,
  unsigned long int xBufferLength,
  const unsigned char* pxSourceBuffer,
  unsigned char* pxTargetBuffer
);
```

Arguments

`xProviderId`	In: The ID of the service provider.
`xBufferLength`	In: The length of the chunk of data to decrypt.
`pxSourceBuffer`	In: A pointer to the buffer that contains the data to decrypt.
`pxTargetBuffer`	In/Out: A pointer to the buffer where to store the decrypted data.

Return value

`CA_COMMAND_ACCEPTED`	If the command could successfully be initiated,
`CA_COMMAND_REJECTED`	otherwise.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.
The message field `INFO_CA_COMMAND` is set to `CA_DECRYPT_BUFFER`. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the decryption of the chunk of data could complete successfully or not, respectively.

See also

Function `caEncryptBuffer`

3.2.3 caEncryptBuffer

Description

Encrypts a chunk of data.

Prototype

```
int caEncryptBuffer
(
  unsigned long int xProviderId,
  unsigned long int xBufferLength,
  const unsigned char* pxSourceBuffer,
  unsigned char* pxTargetBuffer
);
```

Arguments

`xProviderId`	In: The ID of the service provider.
`xBufferLength`	In: The length of the chunk of data to encrypt.
`pxSourceBuffer`	In: A pointer to the buffer that contains the data to encrypt.
`pxTargetBuffer`	In/Out: A pointer to the buffer where to store the encrypted data.

Return value

`CA_COMMAND_ACCEPTED`	if the command could successfully be initiated
`CA_COMMAND_REJECTED`	otherwise.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.
The message field `INFO_CA_COMMAND` is set to `CA_ENCRYPT_BUFFER`. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the encryption of the chunk of data could complete successfully or not, respectively.

See also

Function `caDecryptBuffer`

3.2.4 caGetCurrencyInfo

Description

Gives information about the local currency.

Prototype

```
typedef struct
{
  char symbol[CA_SYMBOL_SIZE+1];
  unsigned long int factor;
} TCaCurrencyInfo;

int caGetCurrencyInfo
(
  TCaCurrencyInfo* pxInfo
);
```

Arguments

`pxInfo`	In: A pointer where to store the information.

Return value

`CA_COMMAND_ACCEPTED`	if the command could successfully be initiated,
`CA_COMMAND_REJECTED`	otherwise.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.
The message field `INFO_CA_COMMAND` is set to `CA_GET_CURRENCY_INFO`. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the retrieval of the information about the local currency could complete successfully or not, respectively. Upon successful completion of the command, the message field `INFO_CA_RESULT` is set to a copy of the pointer `pxInfo` to a `TCaCurrencyInfo` structure.
In this structure, the field `symbol` gives the local currency symbol as an ASCIIZ string, and the field `factor` gives the local currency-scaling factor.

See also

None.

3.2.5 caOpenReturnChannel

Description

Opens a connection on the return channel.

Prototype

```
int caOpenReturnChannel
(
  unsigned long int xProviderId,
  unsigned long int* pxChannelId
);
```

Arguments

`xProviderId`	In: The ID of the service provider.
`pxChannelId`	In/Out: A pointer where to store the ID of the channel.

Return value

`CA_COMMAND_ACCEPTED`	if the command could successfully be initiated, `CA_COMMAND_REJECTED` otherwise.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.
The message field `INFO_CA_COMMAND` is set to `CA_OPEN_RETURN_CHANNEL`. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the connection on the return channel could be opened successfully or not, respectively.

See also

Function `caCloseReturnChannel`
Function `caReadReturnChannel`
Function `caWriteReturnChannel`

3.2.6 caReadReturnChannel

Description

Reads a chunk of data from a connection on the return channel.

Prototype

```
int caReadReturnChannel
(
  unsigned long int xChannelId,
  unsigned long int xBufferLength,
  unsigned char* pxBuffer
);
```

Arguments

`xChannelId`	In: The ID of the channel.
`xBufferLength`	In: The length of the chunk of data to read.
`pxBuffer`	In/Out: A pointer to the buffer where to store the read data.

Return value

`CA_COMMAND_ACCEPTED`	if the command could successfully be initiated,
`CA_COMMAND_REJECTED`	otherwise.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.
The message field `INFO_CA_COMMAND` is set to `CA_READ_RETURN_CHANNEL`. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the chunk of data could be read successfully from the connection on the return channel or not, respectively.

See also

Function `caCloseReturnChannel`
function `caOpenReturnChannel`
function `caWriteReturnChannel`

3.2.7 caWriteReturnChannel

Description

Writes a chunk of data to a connection on the return channel.

Prototype

```
int caWriteReturnChannel
(
  unsigned long int xChannelId,
  unsigned long int xBufferLength,
  const unsigned char* pxBuffer
);
```

Arguments

`xChannelId`	In: The ID of the channel.
`xBufferLength`	In: The length of the chunk of data to write.
`pxBuffer`	In/Out: A pointer to the buffer that contains the data to write.

Return value

`CA_COMMAND_ACCEPTED`	if the command could successfully be initiated,
`CA_COMMAND_REJECTED`	otherwise.

Side effects

None.

Comments

This function is asynchronous and returns right away with a return value that indicates whether the command could successfully be initiated. If so the application receives a message of class `MSG_CLASS_CA` and type `MSG_TYPE_CA_COMMAND` upon completion of the command.
The message field `INFO_CA_COMMAND` is set to `CA_WRITE_RETURN_CHANNEL`. The message field `INFO_CA_STATUS` is set either to `CA_GOOD` or to `CA_BAD` depending on whether the chunk of data could be written successfully to the connection on the return channel or not, respectively.

See also

Function `caCloseReturnChannel`
Function `caOpenReturnChannel`
Function `caReadReturnChannel`