

# **OpenXCAS API Reference Manual**

**API version: 1.5.0**

**Compatible firmware version: over 0.9.1528**

**Date: May 05 2009**

<b>1</b>	<b>API REFERENCES</b>	<b>4</b>
	<code>void openxcas_show_info_and_exit(char * module_name, char * version_info);</code>	4
	<code>int openxcas_open(char * module_name);</code>	4
	<code>int openxcas_open_with_smartcard(char * module_name);</code>	4
	<code>int openxcas_close(void);</code>	4
	<code>void openxcas_debug_message_onoff(int bVerbose);</code>	5
	<code>int openxcas_get_smartcard_device(unsigned int index);</code>	5
	<code>int openxcas_release_smartcard_device(unsigned int index);</code>	5
	<code>char * openxcas_get_working_directory(void);</code>	5
	<code>int openxcas_get_message(openxcas_msg_t * message, int wait);</code>	6
	<code>int openxcas_put_message(int streamd_id, unsigned int sequence, int msg_type, unsigned char *msg_buf, unsigned int msg_size);</code>	6
	<code>int openxcas_send_private_message(int stream_id, unsigned int sequence, int msg_type, unsigned char *msg_buf, unsigned int msg_size);</code>	7
	<code>int openxcas_key_not_found(int stream_id, unsigned int sequence);</code>	7
	<code>int openxcas_start_filter_ex(int stream_id, unsigned int sequence, unsigned short pid, unsigned char * mask, unsigned char * comp, filter_callback callback_func);</code>	8
	<code>int openxcas_stop_filter_ex(int stream_id, unsigned int sequence, int filter_index);</code>	9
	<code>int openxcas_filter_callback_ex(int stream_id, unsigned int sequence, struct stOpenXCAS_Data * openxcas_data);</code>	9
	<code>int openxcas_add_filter(int stream_id, int type, unsigned short ca_system_id, unsigned short target_pid, unsigned short pid, unsigned char * mask, unsigned char * comp, ecmemm_callback callback_func);</code>	10
	<code>int openxcas_start_filter(int stream_id, unsigned int sequence, int type);</code>	11

<code>int openxcas_stop_filter(int stream_id, int type);</code>	11
<code>int openxcas_remove_filter(int stream_id, int type);</code>	11
<code>int openxcas_filter_callback(int stream_id, unsigned int sequence, int type, struct stOpenXCAS_Data * openxcas_data);</code>	12
<code>int openxcas_set_key(int stream_id, unsigned int sequence, unsigned short ca_system_id, unsigned short cipher_index, unsigned char * even, unsigned char * odd);</code>	12
<b>2 HOW TO USE API</b>	<b>14</b>
2.1 The Overview of API	14
2.2 How to descramble the channel	15
2.3 How to get multi-EMM	17
2.4 How to get multi-ECM and descramble the channel	18
<b>3 HOW TO BUILD AND INSTALL OPENXCAS MODULE</b>	<b>20</b>
3.1 Extract toolchain and sample codes	20
3.2 Download the latest HXOpenXCAS_Sample.tgz	20
3.3 Build sample codes	20
3.4 Package sample module	20
3.5 Install sample module	21

## 1 API References

**Be careful! This API is not safe in multi-process**

**void openxcas\_show\_info\_and\_exit(char \* module\_name, char \* version\_info);**

### DESCRIPTION

- This function will be used for checking compatibility with API. After printing information, it will be terminated automatically. Be careful! Module name & version info should be set exactly

**int openxcas\_open(char \* module\_name);**

### DESCRIPTION

- This function opens XCAS interface without smartcard support

### PARAMETER

- **module\_name**: the identifier for generating message queue key.

### RETURN VALUE

- This function returns zero on success. On error -1 is returned.

**int openxcas\_open\_with\_smartcard(char \* module\_name);**

### DESCRIPTION

- This function opens XCAS interface with smartcard support

### PARAMETER

- **module\_name**: the identifier for generating message queue key.

### RETURN VALUE

- This function returns zero on success. On error -1 is returned.

**int openxcas\_close(void);**

### DESCRIPTION

- This function closes XCAS interface

#### **RETURN VALUE**

- This function returns zero on success. On error -1 is returned.

**void openxcas\_debug\_message\_onoff(int bVerbose);**

#### **DESCRIPTION**

- This function turns on/off debug messages. This function should be called after opening openxcas

#### **PARAMETER**

- **bVerbose:** set bVerbose to '1' if you want to enable debug messages

**int openxcas\_get\_smartcard\_device(unsigned int index);**

#### **DESCRIPTION**

- This function gets the authority of smartcard device

#### **PARAMETER**

- **index:** the index of smartcard( in case of AZBOX HD, only index 0 is available)

#### **RETURN VALUE**

- This function returns zero on success. On error -1 is returned.

**int openxcas\_release\_smartcard\_device(unsigned int index);**

#### **DESCRIPTION**

- This function releases the authority of smartcard device

#### **PARAMETER**

- **index:** the index of smartcard( in case of AZBOX HD, only index 0 is available)

#### **RETURN VALUE**

- This function returns zero on success. On error -1 is returned.

**char \* openxcas\_get\_working\_directory(void);**

#### **DESCRIPTION**

- This function is used to get the working directory

## RETURN VALUE

- This function returns full path of working directory.

**int openxcas\_get\_message(openxcas\_msg\_t \* message, int wait);**

## DESCRIPTION

- This function is used to receive message from XCAS or UI

## PARAMETER

- **message:** On success, message should be filled as follows:

```
typedef struct stOpenCASMessage {  
    long mtype;      /* do not touch, used by message queue */  
  
    int stream_id;  
    unsigned int sequence;  
  
    int cmd;  
  
    int buf_len;  
    unsigned char buf[OPENXCAS_MSG_MAX_LEN];  
} openxcas_msg_t;
```

- **wait:** not available

## RETURN VALUE

- On success, this function return 1 or zero if the timeout expires before any interesting happens. On error, -1 is returned.

**int openxcas\_put\_message(int streamd\_id, unsigned int sequence, int msg\_type, unsigned char \*msg\_buf, unsigned int msg\_size);**

## DESCRIPTION

- This function is used to send message to XCAS

## PARAMETER

- **stream\_id:** Channel ID you want to descramble. (As more than 2 channels including LIVE, Recording should be descrambled, each channel ID are automatically assigned in API to classify them)

- **sequence:** An unique value to be used for synchronization between module and XCAS, which increases by 1 in every channel change. Always the sequence got from `openxcas_get_message` is used.
- **msg\_type:** A message type to be delivered to XCAS.
- **msg\_buf:** A message delivered to XCAS.
- **msg\_size:** Size of `msg_buf`

#### RETURN VALUE

- On success, this function return 1 or zero if the timeout expires before anything interesting happens. On error, -1 is returned.

**int openxcas\_send\_private\_message(int stream\_id, unsigned int sequence, int msg\_type, unsigned char \*msg\_buf, unsigned int msg\_size);**

#### DESCRIPTION

- This function is used to send private message to UI.

#### PARAMETER

- **stream\_id:** Channel ID you want to descramble. (As more than 2 channels including LIVE, Recording should be descrambled, each channel ID are automatically assigned in API to classify them)
- **sequence:** An unique value to be used for synchronization between module and XCAS, which increases by 1 in every channel change. Always the sequence got from `openxcas_get_message` is used.
- **msg\_type:** A message type to be delivered to XCAS.
- **msg\_buf:** A message delivered to XCAS.
- **msg\_size:** Size of `msg_buf`

#### RETURN VALUE

- On success, this function return 1 or zero if the timeout expires before any interesting happens. On error, -1 is returned.

**int openxcas\_key\_not\_found(int stream\_id, unsigned int sequence);**

#### DESCRIPTION

- This function is used to inform XCAS that current channel is not descrambled by module. Be careful! If you call this function, all filter information is reset (same to call `openxcas_remove_filter()`).

#### PARAMETER

- **stream\_id**: Channel ID you want to descramble. (As more than 2 channels including LIVE, Recording should be descrambled, each channel ID are automatically assigned in API to classify them)
- **sequence**: An unique value to be used for synchronization between module and XCAS, which increases by 1 in every channel change. Always the sequence got from openxcas\_get\_message is used.

#### RETURN VALUE

- On success, this function return 1 or zero if the timeout expires before anything interesting happens. On error, -1 is returned.

**int openxcas\_start\_filter\_ex(int stream\_id, unsigned int sequence, unsigned short pid, unsigned char \* mask, unsigned char \* comp, filter\_callback callback\_func);**

#### DESCRIPTION

- This function is used to filter PSI section by PID. This can be used to filter the plural numbers of ECM or EMM pid.

#### PARAMETER

- **stream\_id**: Channel ID you want to descramble. (As more than 2 channels including LIVE, Recording should be descrambled, each channel ID are automatically assigned in API to classify them)
- **sequence**: An unique value to be used for synchronization between module and XCAS, which increases by 1 in every channel change. Always the sequence got from openxcas\_get\_message is used.
- **pid**: Configures pid to be filtered.
- **mask**: Configures mask of section to be filtered. (From table\_id, 12 digits can be used.)
- **comp**: Configures result after applying mask value of section to be filtered. (From table\_id, 12 digits can be used.)
- **callback\_func**: The callback function's pointer which will be called when there is any event to a configured filter, and its prototype is as below.

```
typedef void (*filter_callback) (int stream_id, unsigned int sequence, int filter_index, unsigned short pid, unsigned char *pBuf, int size);
```

#### RETURN VALUE

- This function returns filter index on success. On error -1 is returned.



**int openxcas\_stop\_filter\_ex(int stream\_id, unsigned int sequence, int filter\_index);**

#### **DESCRIPTION**

- This function is used to stop PID filters.

#### **PARAMETER**

- **stream\_id**: Channel ID you want to descramble. (As more than 2 channels including LIVE, Recording should be descrambled, each channel ID are automatically assigned in API to classify them)
- **sequence**: An unique value to be used for synchronization between module and XCAS, which increases by 1 in every channel change. Always the sequence got from openxcas\_get\_message is used.
- **filter\_index**: Returned index after calling openxcas\_start\_filter\_ex().

#### **RETURN VALUE**

- This function returns over zero on success. On error -1 is returned.

**int openxcas\_filter\_callback\_ex(int stream\_id, unsigned int sequence, struct stOpenXCAS\_Data \* openxcas\_data);**

#### **DESCRIPTION**

This function is used to call your callback function registered by you. When an event, registered by openxcas\_add\_filter\_ex(), occurs, XCAS API does not call immediately callback function. Instead, send OPENXCAS\_PID\_FILTER\_CALLBACK type message to a module. Once the module receives that message through openxcas\_get\_message() function and it calls the function and data included in that message, a callback function registered by openxcas\_add\_filter\_ex is called.

#### **PARAMETER**

- **stream\_id**: Channel ID you want to descramble. (As more than 2 channels including LIVE, Recording should be descrambled, each channel ID are automatically assigned in API to classify them)
- **sequence**: An unique value to be used for synchronization between module and XCAS, which increases by 1 in every channel change. Always the sequence got from openxcas\_get\_message is used.

#### **RETURN VALUE**

- On success, this function return 1 or zero if the timeout expires before anything interesting happens. On error, -1 is returned.

```
int openxcas_add_filter(int stream_id, int type, unsigned short ca_system_id, unsigned short target_pid, unsigned short pid, unsigned char * mask, unsigned char * comp, ecmemm_callback callback_func);
```

## DESCRIPTION

- This function is used to add PID filters and create cipher to XCAS. To create cipher, type must be created with OPENXCAS\_FILTER\_ECM. If you want to use general pid filter feature, such as EMM filtering, use openxcas\_add\_filter\_ex() function.

## PARAMETER

- **stream\_id**: Channel ID you want to descramble. (As more than 2 channels including LIVE, Recording should be descrambled, each channel ID are automatically assigned in API to classify them)
- **type**: Configures the purpose of filtering, and possible types are as below.

```
enum eOPENXCAS_FILTER_TYPE {  
    OPENXCAS_FILTER_UNKNOWN = 0,  
    OPENXCAS_FILTER_ECM,  
    OPENXCAS_FILTER_EMM, // Not supported  
};
```

- **ca\_system\_id**: CAS system ID selected by module.
- **target\_pid**: If type is OPENXCAS\_FILTER\_ECM, this function is used to select a pid to descramble. If the ecm\_pid of video, audio and data is different, you need to configure target\_pid for each and call this function, and cipher will be created for each. If the ecm\_pid of video, audio and data is same, set as 0xffff and call this function. Then, video, audio and data use the same cipher. target\_pid will not have any meaning to other types except OPENXCAS\_FILTER\_ECM.
- **pid**: Configures pid to be filtered.
- **mask**: Configures mask of section to be filtered. (From table\_id, 12 digits can be used.)
- **comp**: Configures result after applying mask value of section to be filtered. (From table\_id, 12 digits can be used.)
- **callback\_func**: The callback function's pointer which will be called when there is any event to a configured filter, and its prototype is as below.

<pre>typedef void (*ecmemm_callback) (int stream_id, unsigned int sequence, int cipher_index,     unsigned int ca_system_id, unsigned char *pEcm, int Len, unsigned short pid);</pre>
---

## RETURN VALUE

- This function returns zero on success. On error -1 is returned.

**int openxcas\_start\_filter(int stream\_id, unsigned int sequence, int type);**

#### **DESCRIPTION**

- This function is used to start PID filters, and activates filters and cipher created by openxcas\_add\_fillter() function.

#### **PARAMETER**

- **stream\_id**: Channel ID you want to descramble. (As more than 2 channels including LIVE, Recording should be descrambled, each channel ID are automatically assigned in API to classify them)
- **sequence**: An unique value to be used for synchronization between module and XCAS, which increases by 1 in every channel change. Always the sequence got from openxcas\_get\_message should be used.
- **type**: Determines the filter type to start.

#### **RETURN VALUE**

- On success, this function return 1 or zero if the timeout expires before anything interesting happens. On error, -1 is returned.

**int openxcas\_stop\_filter(int stream\_id, int type);**

#### **DESCRIPTION**

- This function is used to stop PID filters.

#### **PARAMETER**

- **stream\_id**: Channel ID you want to descramble. (As more than 2 channels including LIVE, Recording should be descrambled, each channel ID are automatically assigned in API to classify them)
- **type**: Determines the filter type to stop.

#### **RETURN VALUE**

- On success, this function return 1 or zero if the timeout expires before anything interesting happens. On error, -1 is returned.

**int openxcas\_remove\_filter(int stream\_id, int type);**

#### **DESCRIPTION**

- This function is used to remove PID filters.

#### **PARAMETER**

- **stream\_id**: Channel ID you want to descramble. (As more than 2 channels including LIVE, Recording should be descrambled, each channel ID are automatically assigned in API to classify them)

- **type**: Determines the filter type to remove.

#### RETURN VALUE

- This function returns zero on success. On error -1 is returned.

```
int openxcas_filter_callback(int stream_id, unsigned int sequence, int type, struct stOpenXCAS_Data * openxcas_data);
```

#### DESCRIPTION

- This function is used to call your callback function, which you registered. When an event, registered by `openxcas_add_filter()`, occurs, XCAS API does not immediately call callback function. Instead, send below two types of message to a module

OPENXCAS\_ECM\_CALLBACK

OPENXCAS\_EMM\_CALLBACK

Once the module receives the above messages through `openxcas_get_message()` function and it calls the function and data included in that message, a callback function registered by `openxcas_add_filter()` is called.

#### PARAMETER

- **stream\_id**: Channel ID you want to descramble. (As more than 2 channels including LIVE, Recording should be descrambled, each channel ID are automatically assigned in API to classify them)
- **sequence**: An unique value to be used for synchronization between module and XCAS, which increases by 1 in every channel change. Always the sequence got from `openxcas_get_message` should be used.
- **type**: Determines the filter type to start.

#### RETURN VALUE

- This function returns zero on success. On error -1 is returned.

```
int openxcas_set_key(int stream_id, unsigned int sequence, unsigned short ca_system_id, unsigned short cipher_index, unsigned char * even, unsigned char * odd);
```

#### DESCRIPTION

- This function is used to set Control Word to XCAS.

#### PARAMETER

- **stream\_id**: Channel ID you want to descramble. (As more than 2 channels including LIVE, Recording should be descrambled, each channel ID are automatically assigned in API to classify them)
- **sequence**: An unique value to be used for synchronization between module and XCAS, which increases by 1 in every channel change. Always the sequence got from `openxcas_get_message` should be used.
- **ca\_system\_id**: CAS system ID selected by module.
- **cipher\_index**: The index of cipher created in using `openxcas_add_filter()`. Cipher\_index value is used included in ECM callback.
- **even**: Even control word(if there is nothing, it is NULL)
- **odd**: Odd control word(if there is nothing, it is NULL)

#### RETURN VALUE

- On success, this function return 1 or zero if the timeout expires before anything interesting happens. On error, -1 is returned.

## 2 How to use API

This chapter describes examples on how to process messages and how to use APIs in what sequence. For more details, you can refer to the source codes included in OpenXCAS\_Sample.

### 2.1 The Overview of API

OpenXCAS, written in single process base, exchanges command and data between CAS module and STB software through message queue. To write CAS module with API, initialize your environment according to the procedure shown in Fig. 1 and configure a main loop.

#### STEP 1

With `openxcas_get_working_directory()` function, you can get an absolute directory in STB at which the module is installed. All the files, received with `openxcas_get_working_directory()` function, are located in this directory.

#### STEP 2

You can initialize modules with `openxcas_open(CAS_MODULE_NAME)`. And the `CAS_MODULE_NAME` is used to create key of message queue. If you want to use smartcard interface, use `openxcas_open_with_smartcard(CAS_MODULE_NAME)` function and smartcard handle can be received through `openxcas_get_smartcard_device(index)` function.

#### STEP 3

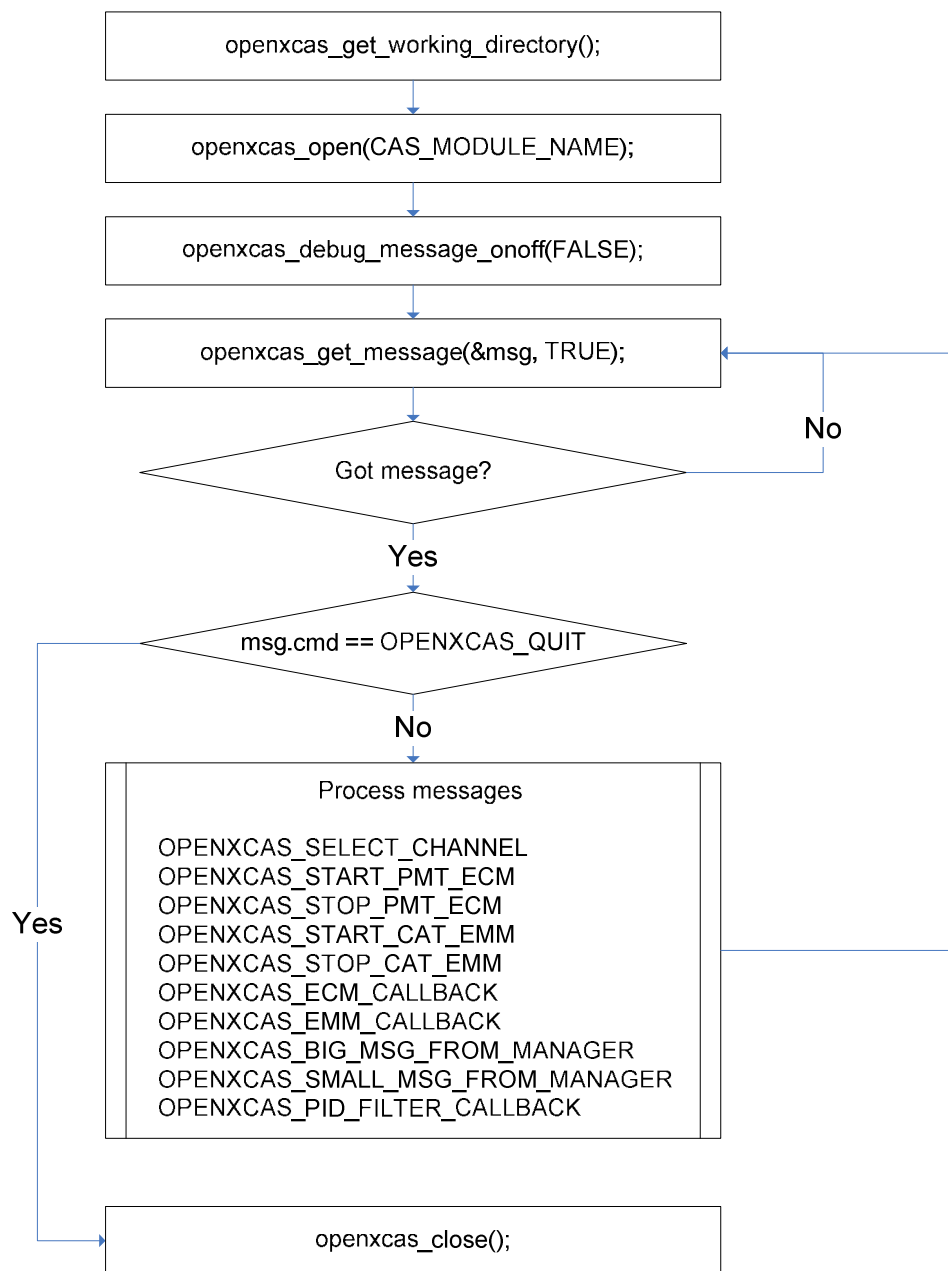
If you want to activate debug messages, you can use `openxcas_debug_message(1)` function.

#### STEP 4

With `openxcas_get_message(&msg, FALSE)`, module waits for message from STB. Specifically, when a user change channels, `OPENXCAS_SELECT_CHANNEL` message with channel information is delivered. If there is previous channel information, `OPENXCAS_STOP_PMT_ECM` message which informs the stop of previous channel is delivered first. Then, `OPENXCAS_START_PMT_ECM` message with PMT information of current channel is delivered. At this time, module should properly process delivered PMT and creates ECM filter and cipher. For the details, you can refer to Section 2.2, 2.3 and 2.4.

#### STEP 5

If the command of message is `OPENXCAS_QUIT`, module calls `openxcas_close()` function and gets terminated.



[Fig. 1] the overview of OpenXCAS API

## 2.2 How to descramble the channel

Fig. 2 depicts the ECM related message flow after channel change.

### STEP 1

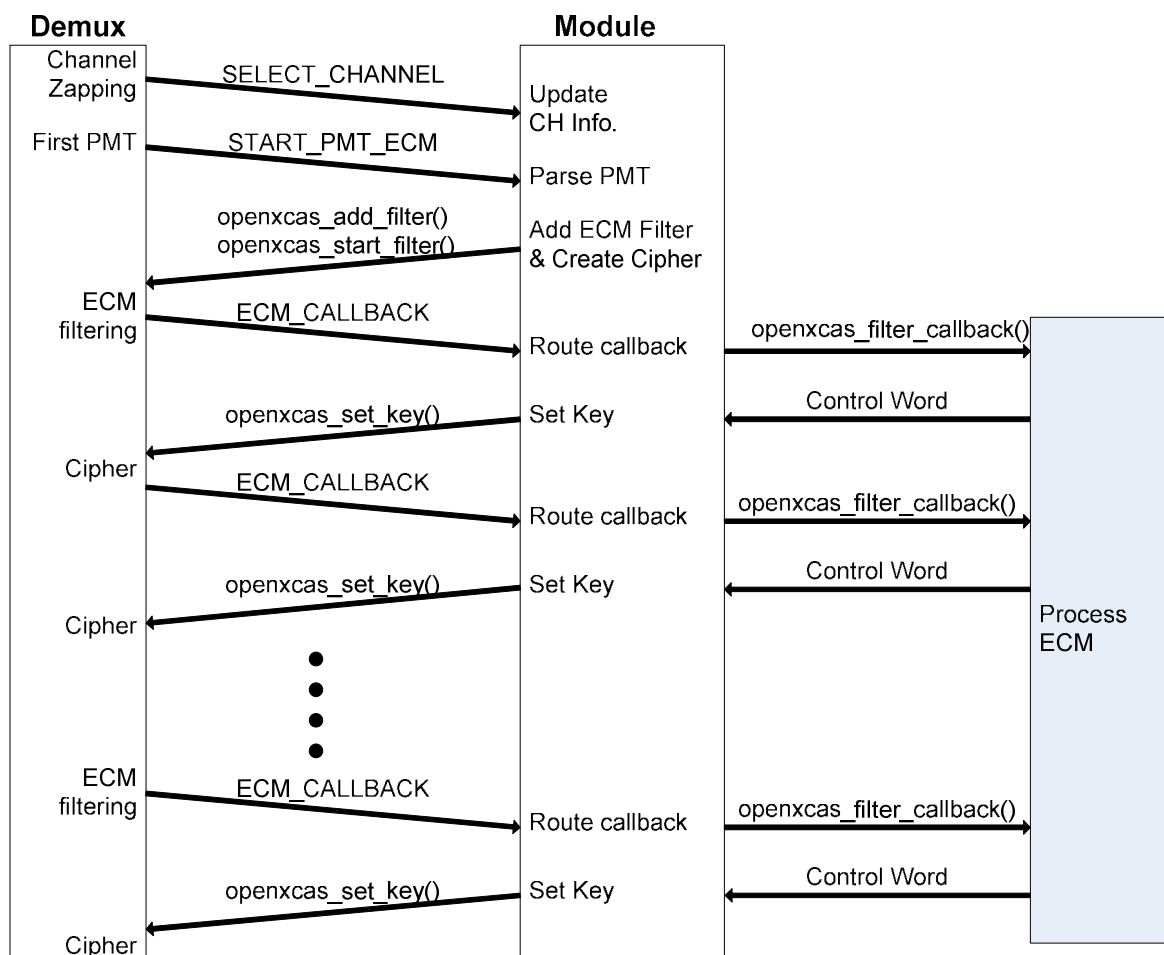
When module receives OPENXCAS\_SELECT\_CHANNEL message, it has to renew the current channel information and wait for PMT information of current channel.

## STEP 2

When module receives OPENXCAS\_PMT\_ECM message, it parses data in the message and seeks ECM pid which is applicable to cas system. Once module finds ECM pid, it creates cipher through penxcas\_add\_filter() and openxcas\_start\_filter() functions, then configures ECM filter.

### STEP 3

When module receives `OPENXCAS_ECM_CALLBACK` message, you have to send the data in `OPENXCAS_ECM_CALLBACK` message to `openxcas_filter_callback()` function. Then, this function will call a callback function which was registered when you called `openxcas_add_filter()`. After properly processing ECM at callback function, module extracts control word, then you can deliver the control word to cipher through `openxcas_set_key()` function.



[Fig. 2] the flowchart of ECM



## 2.3 How to get multi-EMM

Fig. 3 depicts EMM related message flow after channel change.

### STEP 1

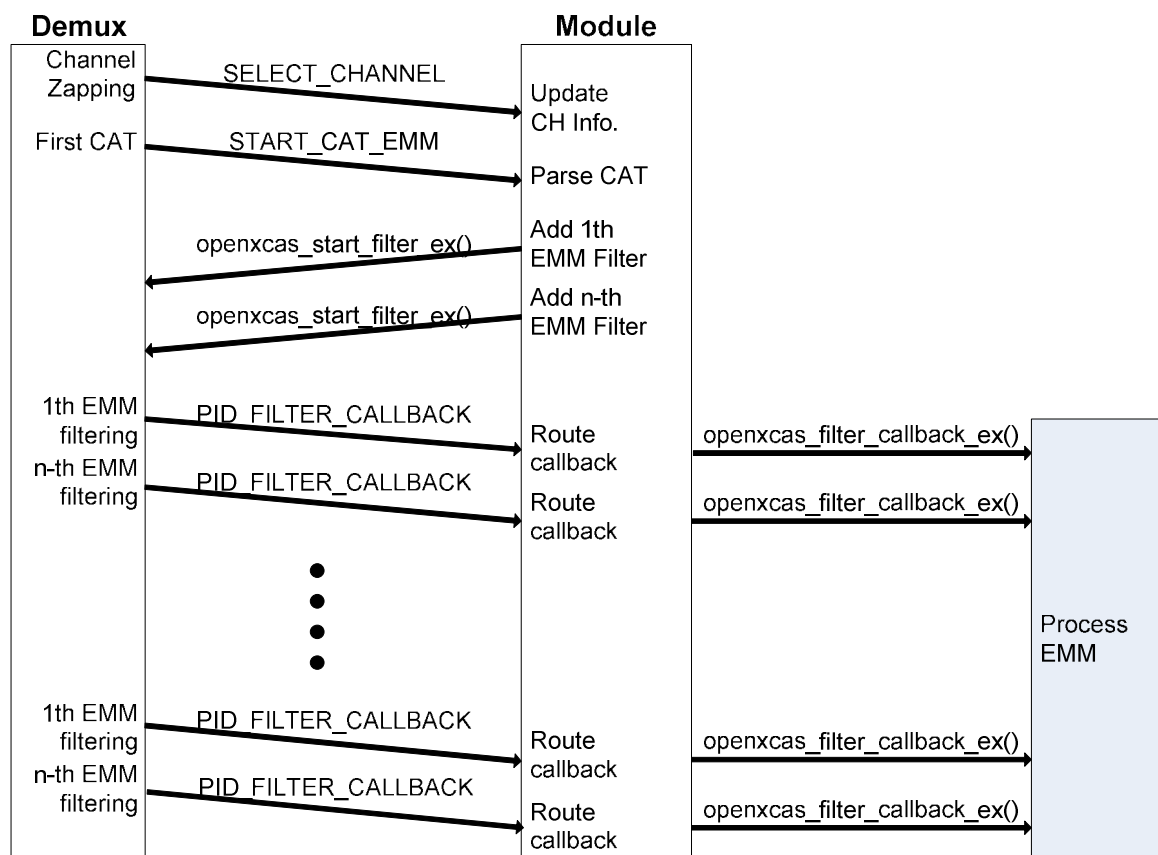
When module receives OPENXCAS\_SELECT\_CHANNEL message, it has to renew the current channel information and wait for CAT information of current channel.

### STEP 2

When module receives OPENXCAS\_CAT\_EMM message, it parses CAT data in the message and seeks EMM pid which is applicable to cas system. And then you can configure EMM filter with openxcas\_start\_filter\_ex() function. Multiple EMM filters can be configured (**Notes: The number of ECM + EMM filters should not exceed 5**).

### STEP 3

When module receives OPENXCAS\_PID\_FILTER\_CALLBACK message, You have to deliver the data in OPENXCAS\_PID\_FILTER\_CALLBACK message to openxcas\_filter\_callback\_ex() function. Then, this function will call a callback function which was registered when you called openxcas\_start\_filter\_ex().



[Fig. 3] the flowchart of EMM

## 2.4 How to get multi-ECM and descramble the channel

Fig. 4 depicts the message flow when you descramble channels with multi-ECM after channel change.

### STEP 1

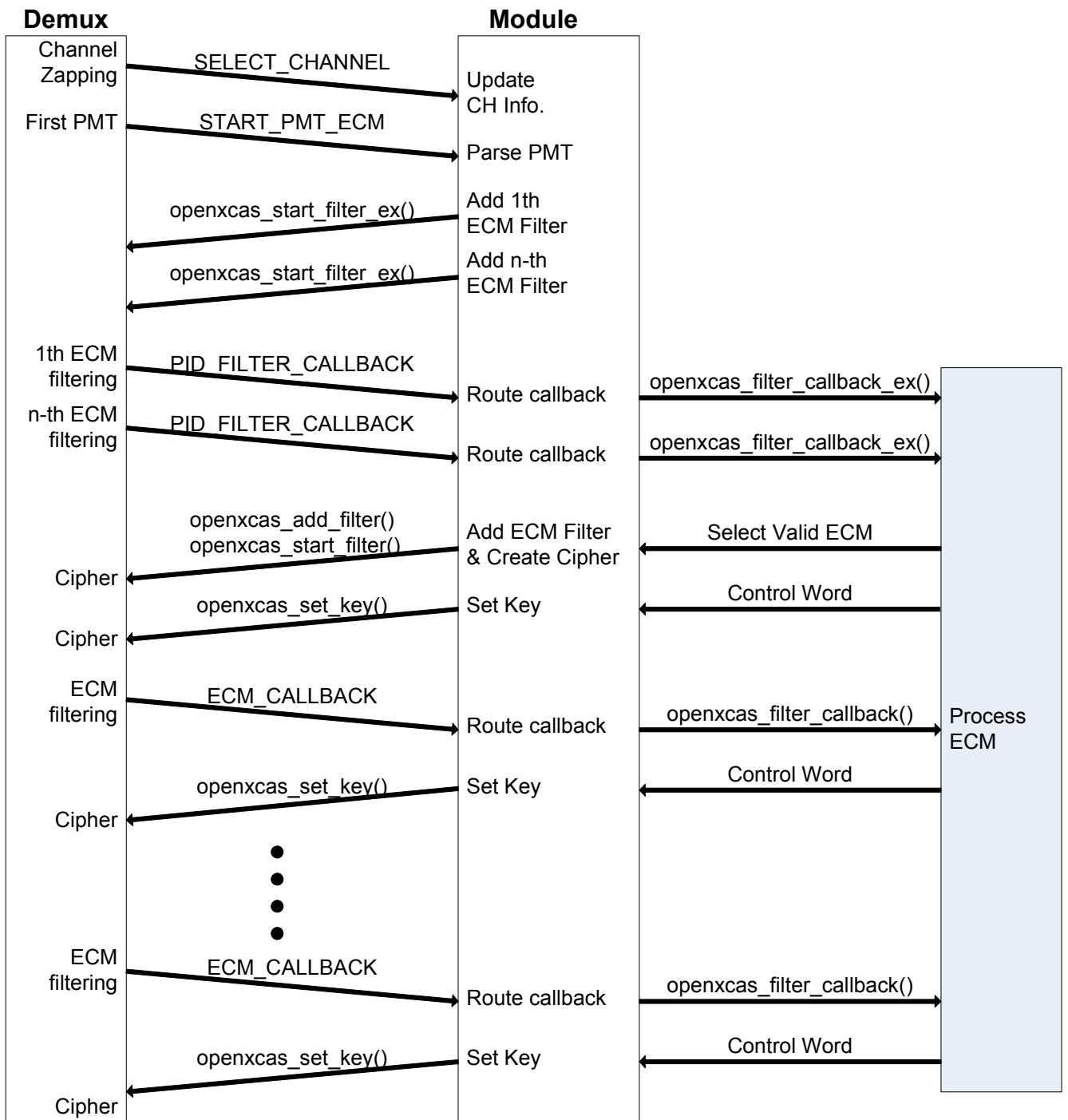
When module receives OPENXCAS\_SELECT\_CHANNEL message, it has to renew the current channel information and wait for PMT information of current channel.

### STEP 2

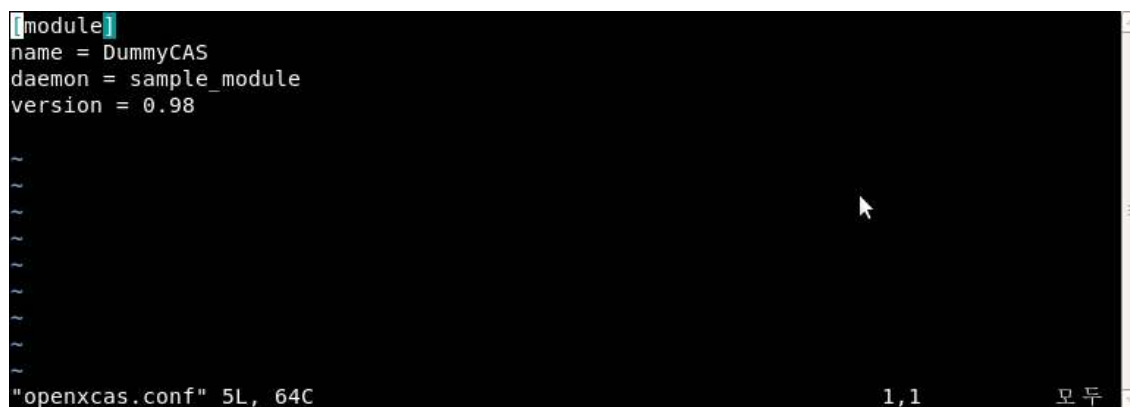
When module receives OPENXCAS\_PMT\_ECM message, it parses PMT data in the message and seeks ECM pid which is applicable to cas system. And then you can configured ECM pid filter with openxcas\_start\_filter\_ex() function. Multiple ECM filters can be configured (**Notes: The number of ECM + EMM filters should not exceed 5**).

### STEP 3

When module receives OPENXCAS\_PID\_FILTER\_CALLBACK message, it has to deliver the data in OPENXCAS\_PID\_FILTER\_CALLBACK message to openxcas\_filter\_callback\_ex() function. Then, this function will call a callback function which was registered when you called openxcas\_start\_filter\_ex(). After selecting supportable ECM at callback function, create cipher and configure ECM filter with supportable ECM pid through openxcas\_add\_filter() and openxcas\_start\_filter() functions. At this process, you must use openxcas\_stop\_filter\_ex() function to close filters which were created by using openxcas\_start\_filter\_ex() function. And if control word was received already, you can create cipher and call openxcas\_set\_key() function to deliver key to cipher immediately.



[Fig. 4] the flowchart of multi ECM



B. Move all files that you need, into specific folder

```
kevin@kevin-laptop:/DATA/Yellow/Working/HXOpenXCAS_Sample/Samples$ ls -al
합계 236
drwxr-xr-x 2 kevin kevin 4096 2009-01-20 19:11 .
drwxr-xr-x 9 kevin kevin 4096 2009-01-20 19:08 ..
-rw-r--r-- 1 kevin kevin 131072 2009-01-20 19:08 Key.bin
-rw-r--r-- 1 kevin kevin 64 2009-01-20 19:08 openxcas.conf
-rwxr-xr-x 1 kevin kevin 88127 2009-01-20 19:08 sample_module
kevin@kevin-laptop:/DATA/Yellow/Working/HXOpenXCAS_Sample/Samples$
```

C. Archive the folder using 'tar czf' as follows:

```
kevin@kevin-laptop:/DATA/Yellow/Working/HXOpenXCAS_Sample$ ls
Key.bin          Samples          sample_common.h  sample_smartcard.c
Makefile         objs.smp8634     sample_module.c   sample_smartcard.h
Makefile.smp8634 objs.x86         sample_module.smp8634 smartcard_drivers
Makefile.x86     openxcas.conf   sample_module.x86
OpenXCASAPI_smp8634 sample_cat.c    sample_pmt.c
OpenXCASAPI_x86  sample_common.c sample_psi.h
kevin@kevin-laptop:/DATA/Yellow/Working/HXOpenXCAS_Sample$ tar czf CAM.tgz Samples/
kevin@kevin-laptop:/DATA/Yellow/Working/HXOpenXCAS_Sample$ ls
CAM.tgz          OpenXCASAPI_x86 sample_common.c    sample_psi.h
Key.bin          Samples          sample_common.h   sample_smartcard.c
Makefile         objs.smp8634     sample_module.c    sample_smartcard.h
Makefile.smp8634 objs.x86         sample_module.smp8634 smartcard_drivers
Makefile.x86     openxcas.conf   sample_module.x86
OpenXCASAPI_smp8634 sample_cat.c    sample_pmt.c
kevin@kevin-laptop:/DATA/Yellow/Working/HXOpenXCAS_Sample$
kevin@kevin-laptop:/DATA/Yellow/Working/HXOpenXCAS_Sample$
```

### 3.5 Install sample module

#### 1) Using terminal

- A. connect to STB using telnet or ssh (account = root, passwd = azbox)
- B. Move CAM.tgz to STB (using ftp, wget, usb and etc)
- C. Run "tar xzf CAM.tgz -C /EMU/OpenXCAS/"
- D. Edit "/EMU/OpenXCAS/module.seq" using 'vi' as follows:

```
MultiCAS
~
~
~
<ain/MtBaekDu/Target/target_td210/EMU/OpenXCAS/module.seq" 1L, 9C 1,1      모두
```

E. Reboot STB using power switch on rear

## 2) Using OpenXCAS Plug-in

Move CAM.tgz to /tmp/Camd.tgz in STB using ftp (account = root, passwd = azbox), and then install as following (Be careful, file name should be 'Camd.tgz')

Step 1. Go to OpenXCAS menu in Plug-Ins



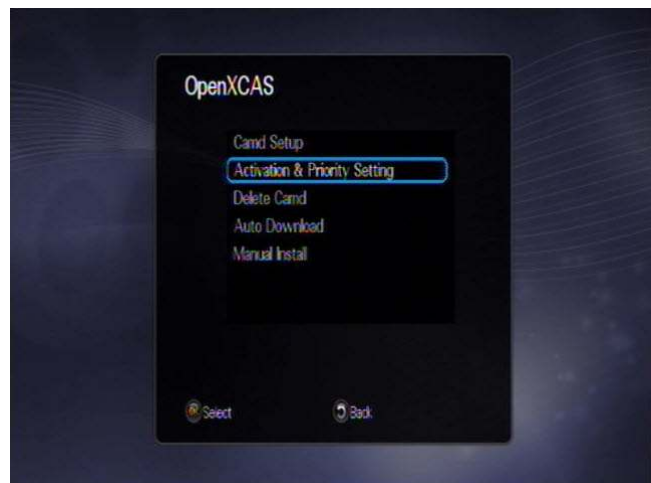
Step 2. Go to 'Manual Install'



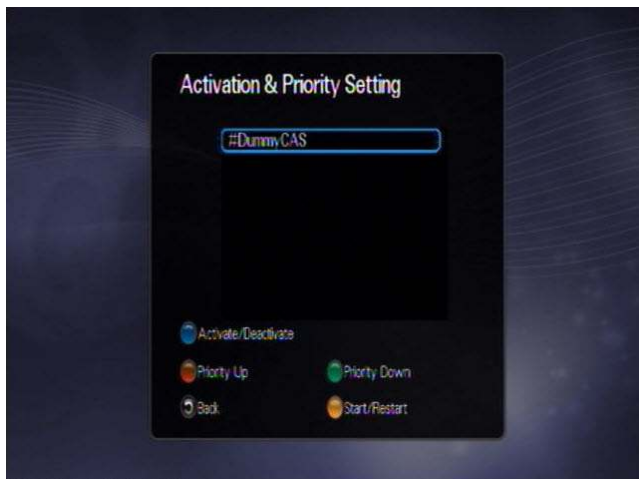
Step 3. Select to 'Install(/tmp/Camd.tgz)'



Step 4. Go to 'Activation & Priority Setting'



Step 5. Activate Module using BLUE key



Step 6. Restart Module using YELLOW key

