TR CAS Porting Specification

[V3.5]User Guide and Reference Manual

20140820
Issue 5
Company Confidential

BEIJING TOPREAL TECHNOLODGY CO. LTD

Table of Contents

Chapter 1 TR CAS Functions

1.1 Introduction

This chapter contains an overview of how the descriptions of the TRCAS functions, contained within Chapters 2 to 9, are presented.

1.2 Function Descriptions

The description of each of function is split into the following areas which detail the name of the function, a list of its parameters, what actions the function should carry out, and a list of rules associated with it.

Description

This provides a basic description of the function.

Prototype

This provides a formal 'C' function prototype for the function, an example of which is shown below.

Prototype

```
S16 TRExample(U16 wParameter1, U8 bParameter2,
                    OUT U8 *pbParameter3)
```

It also provides a detailed list of parameters associated with the function. Each parameter is given a type, a direction and a brief description.

More detailed information regarding the type can be found in *Chapter 10* of this manual, which contains a complete list of all type definitions used by TRCAS.

Parameters

wParameter1 - the first parameter

bParameter2 - the second parameter

pbParameter3 - It is the third parameter, which is a buffer pointer to get data through this function, so its direction is OUT.

Action

This provides a formal list of actions to be carried out by the function.

Returned Values

This provides a list of valid function return values. The return type of most of functions is S16, The typical return values for status are:

- MC_OK, indicating that the function completed successfully.
- MC_NOT_OK, indicating that the function calling is failed.
- MC_PARAMETER_ERROR, indicating that a parameter was passed to the function which violated a parameter restriction.
- MC_NOT_SUPPORTED, indicating the function can not be carried out because it is not supported by the STB.

Called By:

Identifies the caller of the function. This can be either the user's driver layer or MCELL.

Resides In:

Identifies the location of the function. This can be either in user's application or MCELL.

Comments:

Provides any additional comments associated with the function.

Chapter 2 Driver Support

2.1 Introduction

This chapter provides the functional interface to the driver layer for the support of global initialization and getting some states of TRCAS.

2.2 Function Descriptions

2.2.1 MC_CoreInit

Description

This is used when the system booting is complete. This function will call the OS functions to create tasks and semaphores and initialize the whole system of MCELL.

Prototype

S16 MC_CoreInit(S16 iLevel)

Parameters

iLevel - the level of initializing MCELL. Normally it is 0. The special card marriage solution will be used if *iLevel* is set to 1.

Action

Initialize MCELL to start the TRCAS. It will create all CA tasks and semaphores.

Returned Values

0 - Initialization is OK.
1 - Failed to create the task whose Priority is 6.
2 - Failed to initialize smart card.
3 - Failed to initialize ECM driver.
4 - Failed to initialize EMM driver.
5 - Failed to initialize NVRAM driver.

Called By: User's application

Resides In: MCELL

Comments: It must only be called after the system of the box is complete, and only be called once during boot up.

2.2.2 MC_GetRevisionString

Description

This is used to get the version information of MCELL. It can be shown by the OSD of box.

Prototype

U8 *MC_GetRevisionString(void)

Parameters

None

Action

Return the string of the version information. The string buffer size shouldn't be less than 26 bytes.

Returned Values

Return the version string pointer of MCELL.

Called By: User's application

Resides In: MCELL

Comments: None

2.2.3 MC_MNGR_PostChanChangeState

Description

This function is used to inform MCELL that the service is changed.

Prototype

S16 MC_MNGR_PostChanChangeState(U16 wChannelId, U8 bState)

Parameters

wChannelId - The ID of the channel to be changed to.

bState - The relationship state between current service and the service to be changed. The state is 0 if they are in different transponders, and it is 1 if they are in the same transponder.

Action

MCELL will wait for the PMT and CAT table of the new service.

Returned Values

0 - OK

1 - Not OK

Called By: User's application

Resides In: MCELL

Comments: This function must be called after switch channel. The *wChannelId* is just one unique ID of the channel, and application can define it with any characteristic code.

2.2.4 MC_MNGR_PostPsiTable

Description

This function is used to inform MCELL that the PMT or CAT is changed.

Prototype

S16 MC_MNGR_PostPsiTable(CAS_ACTION_EN eAction,
IN PVOID pvTableData)

Parameters

eAction– SI table type (PMT or CAT).

pvTableData – the SI table data pointer.

Action

MCELL will send one message to the task in MCELL to parse the SI table data to get the ECM or EMM PID.

Returned Values

0 - OK

1 - Not OK

Called By: User's application

Resides In: MCELL

Comments: This function must be called after user got the PMT or CAT.

2.2.5 MC_STATE_QueryControl

Description

This function is used to get EMM and ECM control information, for example message, finger print, force play channel, IPPV or IPPT update and so on.

Prototype

CAS_QUERY_STATUS
MC_STATE_QueryControl(CAS_NOTIFY_CONTROL eCtrlType,
U16 wState, OUT PVOID pvParams)

Parameters

eCtrlType - the control information type.

wState - the index to get state.

pvParams - the control information data pointer. The data structure should correspond to the control information type.

Action

Get the relevant control information of the data structure and write to *pvParams*. The used related information types include CAS_MC_FINGER_PRIT, CAS_MC_FORCE_CHANNEL, CAS_MC_IPP_INFO_UPDATE and CAS_MC_NOTIFY_SHORT_MESSAGE. The corresponded data structures are CAS_FINGERPRINT_STRUCT, CAS_FORCECHANNEL_STRUCT, CAS_IPPNOTIFY_STRUCT and CAS_MSG_STRUCT.

Returned Values

0 - CAS_MC_QUERY_SUCCESS

1 - CAS_MC_QUERY_FAIL

2 - CAS_MC_QUERY_NOT_AVAILABLE

Called By: User's application

Resides In: MCELL

Comments: This function must be called by the function which had been registered by MC_STATE_RegisterCaNotify. Please see also: MC_STATE_RegisterCaNotify

2.2.6 MC_STATE_RegisterCaNotify

Description

This function is used to register the TRCAS callback function.

Prototype

S16 MC_STATE_RegisterCaNotify(IN CAS_NOTIFY funcCallback)

Parameters

funcCallback – the callback function entry.

Action

Register the TRCAS callback function, and the notify function will be called when MCELL need to notify information (for example, CA state, message and IPPV/IPPT state) to user application.

Returned Values

0 - OK

1 - Not OK

Called By: User's application

Resides In: MCELL

Comments: It must be called after system initialized complete. Please see also: MC_STATE_QueryControl.

Chapter 3 System Support

3.1 Introduction

This chapter provides the functional interface to system related components.

3.2 Function Descriptions

3.2.1 MC_GetEPurseState

Description

This is used to get the state of the electronic purse which used for IPPV/IPPT, include the slot id, credit and balance.

Prototype

S16 MC_GetEPurseState(OUT CAS_EPURSEINFO_STRUCT *psEPurseInfo,
U8 bIndex)

Parameters

psEPurseInfo - the electronic purse information.

bIndex - the purse index. At present, only support one e-purse,
so the bIndex shold be 0 always.

Action

Read the relevant electronic information from smart card then write into the *psEPurseInfo* buffer.

Returned Values

0 - OK

1 - Not OK

2 - Input parameter error

5 - NOT_SUPPORTED functionality

Called By: User's application

Resides In: MCELL

Comments: None

3.2.2 MC_GetIppProduct

Description

This is used to get the IPPV/IPPT product information, include IPP type (0:IPPV; 1:IPPT), running number, the product entitlement start time and end time.

Prototype

S16 MC_GetIppProduct(OUT CAS_IPPPRODUCT_STRUCT *psProduct, U16 wIndex)

Parameters

psProduct - the IPPV/IPPT product information.

wIndex - the IPPV/IPPT product index saved in the smart card, its value range from 0 to 19.

Action

Read the IPPV/IPPT information from smart card and write into *psProduct*.

Returned Values

0 - OK

1 - Not OK

2 - Input parameter error

3 - No valid product

4 - Nvram error from smart card

5 - NOT_SUPPORTED functionality

Called By: User's application

Resides In: MCELL

Comments: None

3.2.3 MC_GetIppRecord

Description

This is used to get the IPPV/IPPT product consumptive record, include consumptive type, consumptive running number, consumptive sum and consumptive time.

Prototype

S16 MC_GetIppRecord(OUT CAS_IPPRECORD_STRUCT *psRecordData, U16 wIndex)

Parameters

psRecordData - the IPPV/IPPT product consumptive record.

wIndex - the IPPV/IPPT product record index saved in the smart card, its value is from 0 to 99.

Action

Read the IPP product consumptive record from smart card and write into *psRecordData*.

Returned Values

0 - OK

1 - Not OK

2 - Input parameter error

3 - No valid record data

4 - Nvram error from smart card

5 - NOT_SUPPORTED functionality

Called By: User's application

Resides In: MCELL

Comments: None

3.2.4 MC_MC_GetProductType

Description

This is used to get the product type of the current service. The types include PPC and PPV. This information is just for showing information on OSD.

Prototype

S16 MC_GetProductType(OUT U8 *pbType)

Parameters

pbType - the product type. 0 means PPC; 1 means PPV; 0xFF have not got the type.

Action

Read the product type from smart card and return it by *pbType*.

Returned Values

0 - OK

1 - Not OK

Called By: User's application

Resides In: MCELL

Comments: None

3.2.5 MC_GetScMarryState

Description

This function is used to get the binding state between SC and STB. It can be displayed on OSD menu of box.

Prototype

S16 MC_GetScMarryState(OUT char *pcString, U8 bBuffLen);

Parameters

pcString - the smart card number pointer, the string buffer size should be larger than 24 bytes.

bMaxLen - the length of smart card number buffer to be copied.

Action

Read the marry state from CA library. If the returned value is 7, write the smart card number which STB currently marries into *pcString*.

Returned Values

0 - OK

7 - No match between SC and STB. Need to display the smart card number which STB currently marries on the menu of STB.

Other- If it is coming, PLS show it as “Err:code” on the menu of STB.

Called By: User’s application

Resides In: MCELL

Comments: None

3.2.6 MC_GetScSerialNumber

Description

This function is used to get the smart card number. It can be displayed on OSD menu of box.

Prototype

S16 MC_GetScSerialNumber(OUT char *pcString, U8 bMaxLen)

Parameters

pcString - the smart card number pointer, the string buffer size should be larger than 24 bytes.

bMaxLen - the length of smart card number buffer to be copied.

Action

Read the card number from the smart card, and write into *pcString*,

Returned Values

0 - OK

1 - Not OK

Called By: User's application

Resides In: MCELL

Comments: None

3.2.7 MC_GetScStatus

Description

This function is used to get the smart card status.

Prototype

S16 MC_GetScStatus(OUT CAS_SCARD_STATUS *state)

Parameters

State - the smart card state.

0: CAS_SCARD_INSERT; 1: CAS_SCARD_REMOVE;

2: CAS_SCARD_ERROR; 3: CAS_SCARD_UNKNOWN

Action

Read the card state from the smart card and return it by the pointer.

Returned Values

0 - OK

1 - Not OK

Called By: User's application

Resides In: MCELL

Comments: None

3.2.8 MC_GetSmartCardVersion

Description

This is used to get the smart card version. It can be displayed on OSD menu of box.

Prototype

S16 MC_GetSmartCardVersion(OUT char *string, U8 maxLen)

Parameters

string - the string pointer of the smart card version. The buffer size shouldn't be less than 7.

maxLen - the max length of string buffer, it should not be less than 7.

Action

Read the smart card version from smart card then write to string.

Returned Values

0 - OK

1 - Not OK

Called By: User's application

Resides In: MCELL

Comments: None

3.2.9 MC_IppPurchase

Description

This is used to purchase the IPPV/IPPT product.

Prototype

S16 MC_IppPurchase(IN U8 *pbPin, U8 bPinLen,
CAS_IPPPURCHASE_STRUCT sProduct)

Parameters

pbPin - the smart card password pointer.

bPinLen - the smart card password length. Currently, it must be 6.

sProduct - the IPPV/IPPT product information.

Action

Check the validity of the input password. If the password is correct, deduct the cash from the purse and entitle the product.

Returned Values

0 - OK

1 - Not OK

2 - Input parameter error

3 - Input incorrect Pin code

4 - Nvram error from smart card

5 - No enough money

6 - No space for saving product

Called By: User's application

Resides In: MCELL

Comments: None

3.2.10 MC_PostRegionCode

Description

This function is used to control smart card that is only available at given region. If STB gets the right region code, user will get the video/audio or get an error banner "E24 Not allowed in this region".

Prototype

S16 MC_PostRegionCode(U16 wCode)

Parameters

wCode – region code, get it from the descriptor of the first loop of NIT in main frequency.

Action

Input the local region code and get the authority of watching programs. If this card does not belong to this area, “E24 Not allowed in this region” would be shown.

Called By: User’s application

Resides In: MCELL

Comments:

How to get the region_code?

Step 1: Power on STB, it begins to tune the main frequency, and then get the descriptor from the first loop of NIT. Descriptor structure is at the end of this page.

Step 2: Get the frequencies and corresponding region_IDs from the descriptor. Notice: There may be only one region_ID that is not 0 here. If there is a region_ID that is not 0 within this descriptor, it is the right region_id (It indicates that the main frequency is the local main frequency). If each frequency region_ID here is 0, PLS follow step3.

Step 3: Tune above frequencies one by one. If one of them is locked(It is the local main frequency), get the descriptor from the first loop of NIT and get the region_ID.

Descriptor structure

1. In the first layer of the main frequency is NIT table to join the following descriptor:

syntax	bit	description
descriptor_tag	8	0x5f
descriptor_length	8	4*n (Unit is byte)
For(i=0;i<n; i++){		
Frequency	16	Unit is Mhz for cable
Region_ID	16	The value is 0 within main frequency
}		

2. In the first layer of the regional main frequency is NIT table to join the following descriptor：

syntax	bit	description
descriptor_tag	8	0x5f
descriptor_length	8	4
Frequency	16	The value is 0 within region main frequency
Region_ID	16	Post the region_id to MECLL through the function named “MC_PostRegionCode”
}		

3.2.11 MC_ReadFeedDataFromMaster

Description

This function is used to master/slave card. It reads the feed data from master card.

Prototype

S16 MC_ReadFeedDataFromMaster(OUT U8 *pbFeedData,
OUT U8 *pbLen)

Parameters

pbFeedData - the output feed data pointer. The data buffer size must be larger than 128 bytes.

pbLen - the output feed data length.

Action

Read the feed data from master smart card then write to pbFeedData.

Calculate the feed data length and write to pbLen.

Returned Values

0 - OK

1 - MC_NOT_OK, Smart card Error.

3 - MC_DATA_ERROR, Not get useful information.

5 - MC_NOT_SUPPORTED, Smart card attribute error.

6 - MC_STATE_ERROR, Not get useful information.

9 - MC_UNKNOWN_ERROR. Error code is unknown.

Called By: User’s application

Resides In: MCELL

Comments: See also: MC_WriteFeedDataToSlaver.

3.2.12 MC_SCARD_ChangePin

Description

This function is used to change the smart card password.

Prototype

S16 MC_SCARD_ChangePin(IN U8 *pbOldPin, IN U8 *pbNewPin,
U8 bPinLen)

Parameters

pbOldPin - the old password of smart card.

pbNewPin - the new password of smart card.

bPinLen - the length of password, it should be 6.

Action

Check the old password and then replace the old password with the new password.

Returned Values

0 - OK

1 - Not OK

Called By: User's application

Resides In: MCELL

Comments: None

3.2.13 MC_SCARD_GetRating

Description

This function is used to get maturity rating level of the smart card.

Prototype

S16 MC_SCARD_GetRating(OUT U8 *pbRating)

Parameters

pbRating - the rating level of the smart card.

Action

Read the rating level from smart card then write into *pbRating*.

Returned Values

0 - OK

1 - Not OK

Called By: User's application

Resides In: MCELL

Comments: None

3.2.14 MC_SCARD_SetRating

Description

This function is used to set maturity rating level of the smart card.

Prototype

S16 MC_SCARD_SetRating(IN U8 *pbPin, U8 bPinLen, U8 bRating)

Parameters

pbPin - the password of smart card.

bPinLen - the length of the password, it should be 6.

bRating - the rating level of smart card(0 to 10).

Action

Check the password, and then set the rating level of the smart card.

Returned Values

0 - OK

1 - Not OK

Called By: User's application

Resides In: MCELL

Comments: The range of maturity rating level is from 0 to 10.

0 indicates that all services are open, and any service could be descrambled with subscription card.

1 indicates all services are closed, and pin code input is required to descramble on any service.

See also: MC_SCARD_GetRating, MC_SCARD_ParentalCtrlUnlock.

3.2.15 MC_SCARD_ParentalCtrlUnlock

Description

This function is used to check if the input password of maturity rating control is correct.

Prototype

S16 MC_SCARD_ParentalCtrlUnlock(IN U8 *pbPin, U8 bPinLen)

Parameters

pbPin - the input password pointer of the smart card.

bPinLen - the length of the input password, currently it must be 6.

Action

Compare the input password with the password saved in card. If the password is correct, the current service will be descrambled.

Returned Values

0 - OK

1 - Not OK

Called By: User's application

Resides In: MCELL

Comments: None

3.2.16 MC_WriteFeedDataToSlaver

Description

This function is used to write the feed data into slave card.

Prototype

S16 MC_WriteFeedDataToSlaver(IN U8 *pbFeedData, U8 bLen)

Parameters

pbFeedData - the input feed data pointer. The data buffer size must be larger than 128 bytes.

bLen - the input feed data length.

Action

Write the feed data to slave smart card.

Returned Values

0 - OK

1 - Not OK, Smart card error or feed data error.

3 - MC_DATA_ERROR, Feed data error.

4 - MC_MEMORY_RW_ERROR, Write Smart card error.

5 - MC_NOT_SUPPORTED, Smart card attribute error.

6 - MC_STATE_ERROR, Not get the useful feed information.

7 - MC_SCSN_UNMATCHED, Smart card doesn't match with the master.

9 - MC_UNKNOWN_ERROR.

Called By: User's application

Resides In: MCELL

Comments: The function must be called after MC_ReadFeedDataFromMaster. And the calling of MC_ReadFeedDataFromMaster and MC_WriteFeedDataToSlaver must be paired. The attempt to call one function twice continuously will caused failure.

3.2.17 MC_TIME_GetEntitleInfo

Description

This function is used to get the product entitlement information of this smart card for CA menu display and remainder of entitlement expiration.

Prototype

S16 MC_TIME_GetEntitleInfo (OUT U8 *pbStart, OUT U8 *pbEnd,
OUT U32 *pulRemain, U16 wProduct)

Parameters

pbStart - the start time pointer of one product.
pbEnd - the end time pointer of one product.
pulRemain - the remained working day of one product.
wProduct - the product index to be read, the value is from 0 to 319.

Action

Get the relevant entitlement information of product from smart card.

Returned Values

0 - OK
1 - Parameters error.
2 - No entitle info.
3 - Valid entitle, but limited.
4 - Entitle closed.
5 - Entitle paused.

Called By: User's application

Resides In: MCELL

Comments:

1) The time of start and end includes year, month, day, hour, minute and second, and their buffer size must be larger than 7 bytes. For the time format, please refer the STRUCT definition of CAS_TIMESTAMP in **Chapter 11**.
2) On the menu of operator information, the product ID should be from 1 to 320, so CAS integrated engineer should keep correlative between menu and wProduct.

Chapter 4 Kernel

4.1 Introduction

This chapter lists the functional interface to the OS kernel driver. MCELL expects a priority based pre-emptive Operating System.

4.2 Function Descriptions

4.2.1 TRDRV_OS_AllocateMemory

Description

This function is used to allocate memory from the heap. A NULL pointer must be returned if no enough memory of requested size can be allocated or if the requested size is zero.

Prototype

PVOID TRDRV_OS_AllocateMemory(U32 ulSize)

Parameters

ulSize - The memory size .

Action

RTOS will allocate memory used by TRCA.

Returned Values

The allocated memory pointer or NULL if no memory could be allocated.

Called By: MCELL

Resides In: User's driver

Comments: None.

4.2.2 TRDRV_OS_FreeMemory

Description

This function is called to release the previously allocated memory. A release of a NULL pointer must be safe.

Prototype

void TRDRV_OS_FreeMemory(IN PVOID pvMemory)

Parameters

pvMemory - The previously allocated memory pointer.

Action

RTOS will free the allocated memory used by MCELL.

Returned Values

None

Called By: MCELL

Resides In: User's driver

Comments: None

4.2.3 TRDRV_OS_CreateTask

Description

This function will create a task used by MCELL. All created tasks must be able to receive or send message mutually, and all messages use same memory to pass a pointer to data.

Prototype

U32 TRDRV_OS_CreateTask(IN void(*task)(void *arg), S16 iPriority,
U32 ulStackSize)

Parameters

task - The task function entry. The parameter is optional, currently not used.

iPriority - Priority of the task to be created.

ulStackSize - Size of stack required of the task

Action

RTOS will create relevant task used by TRCA.

Returned Values

Return the task identifier.

Called By: MCELL

Resides In: User' driver

Comments: Task priority is relative value. The highest priority is 15, and the lowest is 0. User can map the input priority value into his own OS system.

4.2.4 TRDRV_OS_DelayTask

Description

This function is used to suspend a task for the input period time. And the time of one unit is 1 millisecond.

Prototype

void TRDRV_OS_DelayTask(U32 ulDelayTime)

Parameters

ulDelayTime - The request delay time(ms).

Action

The task will suspend relevant time.

Returned Values

None

Called By: MCELL

Resides In: User's driver

Comments: None.

4.2.5 TRDRV_OS_SendMessage

Description

This function is used to send message to a given task.

Prototype

S16 TRDRV_OS_SendMessage(U32 tTaskId,
IN CAS_OS_MESSAGE *psMsg)

Parameters

tTaskId - Task identifier of the destination task.

psMsg - Message data pointer.

Action

RTOS will send an inter-task type message.

Returned Values

0 - Successful

1 - Failed

Called By: MCELL

Resides In: User's driver

Comments: None.

4.2.6 TRDRV_OS_ReceiveMessage

Description

This function will receive a message from one given task. The relevant task should be blocked while waiting for a message.

Prototype

CAS_OS_MESSAGE *TRDRV_OS_ReceiveMessage(U32 tTaskId)

Parameters

tTaskId - Task identifier.

Action

RTOS will receive the message from a task.

Returned Values

Return the pointer to CAS_OS_MESSAGE memory.

Called By: MCELL

Resides In: User's driver

Comments: None.

4.2.7 TRDRV_OS_CreateSemaphore

Description

This function allows a semaphore to be created and initialized.

Prototype

U32 TRDRV_OS_CreateSemaphore(U32 ulInitialCount)

Parameters

ulInitialCount - initial value of the created semaphore. 0 indicates not available; >0 indicates available.

Action

Create a semaphore with the initial value.

Returned Values

The semaphore identifier

Called By: MCELL

Resides In: User's driver

Comments: None.

4.2.8 TRDRV_OS_SignalSemaphore

Description

This function is used to signals the semaphore, it increment the counter by one.

Prototype

void TRDRV_OS_SignalSemaphore(U32 ulSemaphore)

Parameters

ulSemaphore - The semaphore identifier.

Action

RTOS will signal the semaphore, and make it available.

Returned Values

None

Called By: MCELL

Resides In: User's driver

Comments: None

4.2.9 TRDRV_OS_WaitSemaphore

Description

This function will wait for the semaphore to be signaled. If the semaphore is available, decrement the counter by one.

Prototype

void TRDRV_OS_WaitSemaphore (U32 ulSemaphore)

Parameters

ulSemaphore - The semaphore identifier.

Action

The task calling this function will wait for the semaphore to be signaled.

Returned Values

None

Called By: MCELL

Resides In: User's driver

Comments: The task that calls this function will be blocked while waiting for this semaphore.

Chapter 5 SMART CARD

5.1 Introduction

This chapter lists API between MCELL and the smart card driver. The smart card of TRCAS is compatible to ISO/IEC 7816-3, and uses T=0 protocol.

5.2 Function Descriptions

5.2.1 TRDRV_SCARD_Initialise

Description

This function is called once after system boot up to initialize the driver.

Prototype

S16 TRDRV_SCARD_Initialise(IN S16 (*SC_ATR_Notify)(U8 bCardNumber, U8 bStatus, U8 *pbATR, U8 bProtocol))

Parameters

SC_ATR_Notify - a notify function pointer. It must be called when smart card is inserted or removed. Please set ATR data to 0 if the card is removed.

Action

Initialize the smart card driver and register one notify function into user's driver.

Returned Values

0 - Successful

1 - Failed

Called By: MCELL

Resides In: User's driver

Comments: The *bCardNumber* parameter in notify function is to identify the smart cards in two slots box. It should 0 if only one slot on the box. The *bStatus* indicates the IN or OUT status of the card. 0 is IN and 1 is OUT. The *bProtocol* parameter should be 0 if T=0 protocol used, and 14 if T=14 protocol used.

5.2.2 TRDRV_SCARD_ResetCard

Description

This function is used to reset the smart card, and then the user's driver must call SC_ATR_Notify(See 5.2.1) to notify MCELL the card status and ATR sequence.

Prototype

void TRDRV_SCARD_ResetCard(U8 bCardNumber)

Parameters

bCardNumber - The smart card number of the card to reset.

Action

Reset the smart card.

Returned Values

None

Called By: MCELL

Resides In: User's driver

Comments: Normally, the reset of the smart card should be done by user's application, and then notify the MCELL by SC_ATR_Notify function.

The TRDRV_SCARD_ResetCard function will be called when MCELL thinks the re-reset card is necessary.

5.2.3 TRDRV_SCARD_AdpuFunction

Description

This function is used to transfer data to smart card and return the response data from smart card for t=0 protocol.

Prototype

S16 TRDRV_SCARD_AdpuFunction(U8 bCardNumber, IN U8 *pbTxData,
U32 ulTxLength, OUT U8 *pbRxData,
OUT U32 *pulRxLength)

Parameters

bCardNumber - The number of the smart card to be sent data.

pbTxData - Pointer of the data buffer to send to smart card. This data include T=0 “command header” and the data to be sent.

ulTxLength - The length of the data to be sent to smart card (inclusive the 5 bytes of command header).

pbRxData - Pointer of the received data returned from smart card (inclusive the SW1 and SW2).

pulRxLength - The total length of the data return from smart card. It includes the 2 bytes of SW1 and SW2.

Action

Send 7816 ADPU to smart card and get returned data and SW from the smart card.

Returned Values

0 - Successful

1 - Failed

Called By: MCELL

Resides In: User’s driver

Comments: None.

5.2.4 TRDRV_SCARD_GetSlotState

Description

This function is used to get the inserted/removed state of the smart card in one card slot.

Prototype

```
S16     TRDRV_SCARD_GetSlotState(U8 bCardNumber,
                                 OUT U8 *pbSlotState)
```

Parameters

bCardNumber - The smart card number to get state.

pbSlotState - The pointer of the returned state. 0 is inserted, and 1 is removed.

Action

Get the inserted/removed state in the card slot.

Returned Values

0 - Successful

1 - Failed to get state

Called By: MCELL

Resides In: User' driver

Comments: None.

Chapter 6 DEMUX DRIVER INTERFACE

6.1 Introduction

This chapter lists the functional interface to the MPEG demultiplexer and section filter functions.

6.2 Function Descriptions

6.2.1 TRDRV_DEMUX_Initialise

Description

This function is used to initialize the DEMUX driver. It will be called once in MCELL.

Prototype

S16 TRDRV_DEMUX_Initialise(void)

Parameters

None

Action

Initialize the DEMUX driver.

Returned Values

0 - Successful

1 - Failed

Called By: MCELL

Resides In: User's driver

Comments: None.

6.2.2 TRDRV_DEMUX_AllocateSectionChannel

Description

This function will allocate a section channel to filter data from mpeg TS.

Prototype

```
U32     TRDRV_DEMUX_AllocateSectionChannel(U16 wDemuxId,
                                           U16 wMaxFilterNumber,
                                           U16 wMaxFilterSize,
                                           U32 ulBufferSize)
```

Parameters

wDemuxId –Normally, it could be ignored.

The unique ID Indicates the input source on multiple clients system(for example Dual tuner PVR or multiple tuners input box). It is used to descramble multiple services in the meantime with one smart card. Usually, it could be ignored (set to 0) for the system only need to descramble one service

wMaxFilterNumber - Maximum number of the section filters on this channel.

wMaxFilterSize - Maximum number of the match and mask bytes in the filter.

It indicates the depth of the filter.

ulBufferSize - The buffer size of the channel.

Action

Allocate one section channel to get ECM or EMM data from TS.

Returned Values

The channel number will be returned. It is one identifier of this channel.

The value is from 0 to CAS_DEMUX_INVALID_CHANNEL_ID (*see chapter 10*).

Called By: MCELL

Resides In: User's driver

Comments: 9 channels should be allocated and each channel needs 8 filters.

6.2.3 TRDRV_DEMUX_SetChannelPid

Description

This function will set PID to a section channel which has been allocated. The channel should be disabled before calling the TRDRV_DEMUX_ControlChannel.

Prototype

```
void    TRDRV_DEMUX_SetChannelPid(U32 ulChannelId,
                                  U16 wChannelPid)
```

Parameters

ulChannelId - The identifier of the channel to allocate PID.

wChannelPid - The PID value.

Action

Set the filter PID of one allocated channel.

Returned Values

None

Called By: MCELL

Resides In: User's driver

Comments: None.

6.2.4 TRDRV_DEMUX_RegisterChannelUpcallFct

Description

This function will associate a given channel with one up-call function, which will be called when the channel gets section data. The given channel identifier will be a parameter of the up-call function.

Prototype

void TRDRV_DEMUX_RegisterChannelUpcallFct(U32 ulChannelId,
IN void (*chUpcallFct)(U32 ulChannelId))

Parameters

ulChannelId - The channel identifier of the allocated channel that must register the up-call function.

chUpcallFct - The function to be called when the channel gets data.

Action

Associate an up-call function with one allocated section channel. When user's driver notifies MCELL that there is a section, MCELL will use the function TRDRV_DEMUX_GetSection to read the section data.

Returned Values

The channel number will be returned. It is one identifier of this channel.

The value is from 0 to CAS_DEMUX_INVALID_CHANNEL_ID (*see chapter 10*).

Called By: MCELL

Resides In: User's driver

Comments: None.

6.2.5 TRDRV_DEMUX_AllocateFilter

Description

This function will allocate a section filter on one given channel.

Prototype

U32 TRDRV_DEMUX_AllocateFilter(U32 ulChannelId)

Parameters

ulChannelId - The channel identifier to be allocate filter.

Action

Allocate one section filter on the given channel.

Multiple section filters can be allocated on one channel.

Returned Values

Return the identifier of the filter if it is successful to allocate.

Called By: MCELL

Resides In: User's driver

Comments: None.

6.2.6 TRDRV_DEMUX_SetFilter

Description

This function will set filter mask and match bytes of one given filter. The number of mask and match bytes depends on the *wMaxFilterSize* parameter of TRDRV_DEMUX_AllocateSectionChannel.

Prototype

S16 TRDRV_DEMUX_SetFilter(U32 ulChannelId, U32 ulFilterId,
IN U8 *pbMask, IN U8 *pbData)

Parameters

ulChannelId - The identifier of the channel associated with this filter.

ulFilterId - The identifier of the filter to set the mask and match bytes.

pbMask - The pointer of the mask bytes.

pbData - The pointer of the match bytes.

Action

Set the mask and match bytes of one given filter on a channel.

Returned Values

0 - Successful

1 - Failed

Called By: MCELL

Resides In: User's driver

Comments: The mask and match data of the filter start at table ID, and include section length bytes and section data.

6.2.7 TRDRV_DEMUX_ControlChannel

Description

This function will control a section channel. The state of the channel includes enable, disable and reset.

Prototype

```
void    TRDRV_DEMUX_ControlChannel(U32 ulChannelId,
                                   CAS_DEMUX_CTRL eAction)
```

Parameters

ulChannelId - The identifier of the channel to be controlled.

eAction - The input control type.

Action

Control one given channel. All allocated filters on this channel will have the same status.

Returned Values

None

Called By: MCELL

Resides In: User's driver

Comments: None.

6.2.8 TRDRV_DEMUX_FreeSectionChannel

Description

This function will free one section channel. It will be called after calling TRDRV_DEMUX_FreeFilter.

Prototype

void TRDRV_DEMUX_FreeSectionChannel(U32 ulChannelId)

Parameters

ulChannelId - The identifier of the channel to be released.

Action

Release one section channel.

Returned Values

None

Called By: MCELL

Resides In: User's driver

Comments: None.

6.2.9 TRDRV_DEMUX_FreeFilter

Description

This function will free a section filter on the given channel.

Prototype

void TRDRV_DEMUX_FreeFilter(IN U32 ulChannelId, IN U32 ulFilterId)

Parameters

ulChannelId - The identifier of the channel associated with this filter.

ulFilterId - The identifier of the filter to be released.

Action

Release one filter on the given channel.

Returned Values

None

Called By: MCELL

Resides In: User's driver

Comments: None.

6.2.10 TRDRV_DEMUX_GetSectionData

Description

This function will return a pointer of the section data buffer of the given channel.

Prototype

U8 *TRDRV_DEMUX_GetSectionData(U32 ulChannelId,
OUT U32 *pulSectionSize)

Parameters

ulChannelId - The identifier of the channel to get section data.

pulSectionSize - A pointer of the buffer length of section data.

Action

Return the buffer pointer of the section data to be read.

Returned Values

Pointer to the buffer of section data, and it is NULL if no section data.

Called By: MCELL

Resides In: User's driver

Comments: The owner of the returned pointer is the user's driver. MCELL will call TRDRV_DEMUX_FreeSection to release the pointer.

6.2.11 TRDRV_DEMUX_FreeSectionData

Description

This function will release the buffer pointer of the section channel, and allow new section data enter the buffer. This function will always be called after calling TRDRV_DEMUX_GetSectionData.

Prototype

```
void    TRDRV_DEMUX_FreeSectionData(U32 ulChannelId,
                                    U32 ulSectionSize)
```

Parameters

ulChannelId - The identifier of the channel to release pointer.

ulSectionSize - The section length to be released.

Action

Update the buffer pointer on a section channel.

Returned Values

None

Called By: MCELL

Resides In: User's driver

Comments: None.

Chapter 7 Descrambler

7.1 Introduction

This chapter lists the functional interface to the DVB descrambler functions.

7.2 Function Descriptions

7.2.1 TRDRV_DESC_OpenDescrambler

Description

This function is used to open a descrambler on one demux.

Prototype

U32 TRDRV_DESC_OpenDescrambler(U16 wDemuxId)

Parameters

wDemuxId – Normally, it could be ignored.

The unique ID Indicates the input source to open descrambler on multiple clients system (for example, Dual tuner PVR or multiple tuners input box). It is used to descramble multiple services in the meantime with one smart card. Usually, it could be ignored (set to 0) for the system only need to descramble one service.

Action

Open one descrambler.

Returned Values

Return descrambler identifier or CAS_DEMUX_INVALID_DESCRAMBLER_ID if error.

Called By: MCELL

Resides In: User's driver

Comments: None.

7.2.2 TRDRV_DESC_CloseDescrambler

Description

This function is used to close a descrambler opened before.

Prototype

void TRDRV_DESC_CloseDescrambler(U32 ulDescId)

Parameters

ulDescId - The identifier of the descrambler to be closed.

Action

Close one opened descrambler.

Returned Values

None

Called By: MCELL

Resides In: User's driver

Comments: None.

7.2.3 TRDRV_DESC_SetDescramblerPid

Description

This function is used to associate one PID with one descrambler.

Prototype

void TRDRV_DESC_SetDescramblerPid(U32 ulDescId, U16 wPid)

Parameters

ulDescId - Descrambler identifier to set PID.

wPid - The PID to descramble.

Action

Set a PID on one descrambler.

Returned Values

None

Called By: MCELL

Resides In: User's driver

Comments: None

7.2.4 TRDRV_DESC_SetDescramblerEvenKey

Description

This function is used to set the even keys on a descrambler.

Prototype

void TRDRV_DESC_SetDescramblerEvenKey(U32 ulDescId,
IN U8 *pbEvenKey,
U8 bEvenLen)

Parameters

ulDescId - The descrambler identifier to set CW keys.

pbEvenKey - Pointer of the even keys.

bEvenLen - The length of the keys buffer, and it must be 8 bytes.

Action

Set the even keys on a descrambler.

Returned Values

None

Called By: MCELL

Resides In: User's driver

Comments: None.

7.2.5 TRDRV_DESC_SetDescramblerOddKey

Description

This function is used to set the odd keys on a descrambler.

Prototype

void TRDRV_DESC_SetDescramblerOddKey(U32 ulDescId,
IN U8 *pbOddKey,
U8 bOddLen)

Parameters

ulDescId - The descrambler identifier to set CW keys.

pbOddKey - Pointer of the odd keys.

bOddLen - The length of the keys buffer, and it must be 8 bytes.

Action

Set the odd keys on a descrambler.

Returned Values

None

Called By: MCELL

Resides In: User's driver

Comments: None.

Chapter 8 NVRAM

8.1 Introduction

This chapter lists the functional interface to the NVRAM driver module. The minimal NVRAM size requested by MCELL is 256 bytes.

8.2 Function Descriptions

8.2.1 TRDRV_NVRAM_Initialise

Description

This function is used to initialize the NVRAM driver. It will be called once after in MCELL.

Prototype

S16 TRDRV_NVRAM_Initialise(void)

Parameters

None

Action

Initialize the NVRAM driver.

Returned Values

0 - Successful

1 - Failed

Called By: MCELL

Resides In: User's driver

Comments: None.

8.2.2 TRDRV_NVRAM_Read

Description

This function allows the MCELL read access to the NVRAM.

Prototype

S16 TRDRV_NVRAM_Read(U32 ulOffset, OUT U8 *pbData, U16 wLength)

Parameters

ulOffset - Offset from the start of NVRAM.

pbData - Memory location to which read data should be copied.

wLength - Number of bytes to read.

Action

Read *wLength* bytes from NVRAM starting from location *ulOffset* and copy them into memory starting at address *pbData*.

Returned Values

0 - Successful

1 - Failed

Called By: MCELL

Resides In: User's driver

Comments: None.

8.2.2 TRDRV_NVRAM_Write

Description

This function allows the MCELL write access to the NVRAM.

Prototype

S16 TRDRV_NVRAM_Write(U32 ulOffset, IN U8 *pbData, U16 wLength)

Parameters

ulOffset - Offset from the start of NVRAM.

pbData - Memory location to which data should be copied.

wLength - Number of bytes to write.

Action

Write *wLength* bytes starting at address *pbData* to NVRAM starting at location *ulOffset*. Read back data from NVRAM and compare against data written. If failed to write, return MC_MEMORY_RW_ERROR.

Returned Values

0 - Successful

!0 - Failed

Called By: MCELL

Resides In: User's driver

Comments: None.

Chapter 9 Debug

9.1 Introduction

This chapter lists the functional interface to the debug driver module. It just prints information on display device for checking if the MCELL works well.

9.2 Function Descriptions

9.2.1 MC_Printf

Description

This function prints message on display monitor. Its functionality is same as the *printf* of ANSI C.

Prototype

void MC_Printf(char *format, ...)

Parameters

Format - The format string to display.

Action

Print out message.

Returned Values

None

Called By: MCELL

Resides In: User's driver

Comments: None.

Chapter 10 Screen Display

10.1 Introduction

This Chapter gives screen display requirements of TRCA porting on a specific STB platform.

10.2 CA Error Codes and Explanations

Topreal CA specifies a set of general error codes and associated messages representing errors that may occur in a STB. Those messages should be clearly visible on the display device (e.g. TV).
It is always required to display the error codes and corresponding explanations (e.g. E04) on device. The explanations may be in the language of the country where the STB is applied. The following defines the standard screen text. If operator wishes to display a different screen text, you should inform Topreal tester of it. All messages are to be displayed clearly.
“EXX text string” should be displayed on the screen.

The following lists all possible error codes and text explanations:

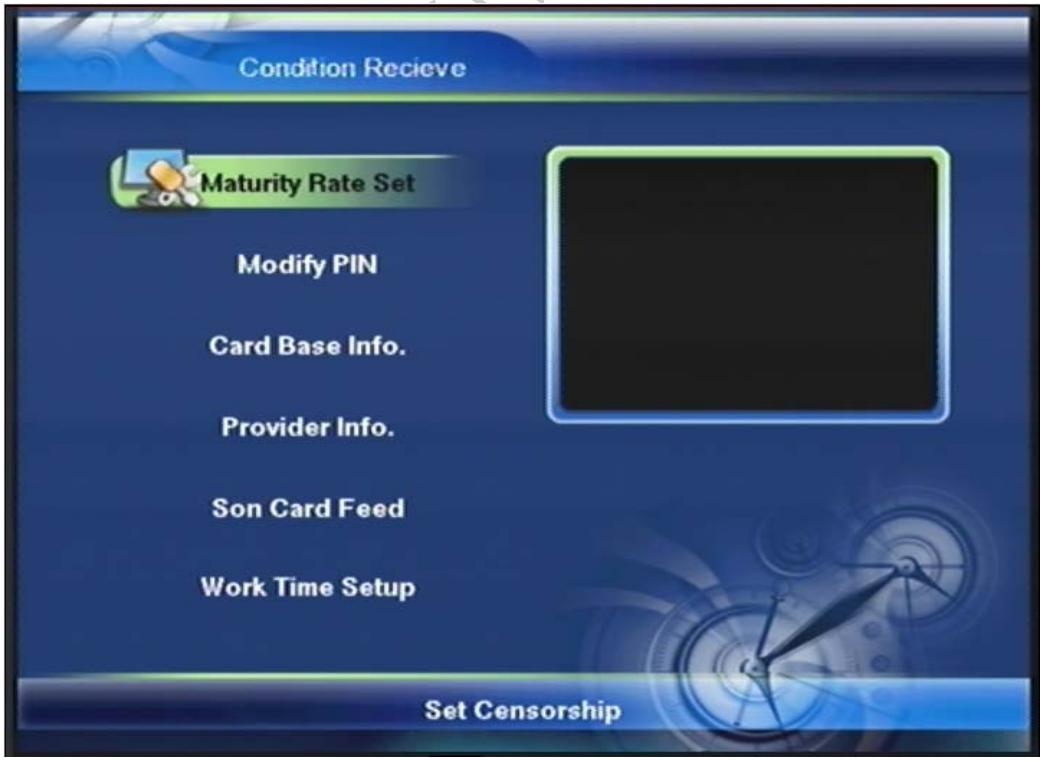
"E00 Service not scrambled",	“E00 当前节目没有加密”,
"E01 ",	“E01 ",
"E02 CA module EEPROM failure",	“E02 CA 模块 EEPROM 失败",
"E03 CA Module failure",	“E03 CA 模块失败”,
"E04 Please insert smart card",	“E04 请插入智能卡”,
"E05 Unknown smart card",	“E05 系统不识别此卡”,
"E06 Smart card failure",	“E06 智能卡失败”,
"E07 Please check smart card",	“E07 请检查智能卡”,
"E08 ",	“E08 ”,
"E09 Smart card EEPROM failure",	“E09 智能卡 EEPROM 失败”,
"E10 ",	“E10 ”,
"E11 Card marriage unmatched",	“E11 机卡不匹配”,
"E12 Please feed son card",	“ E12 请喂养子卡”,
"E13 No service available",	“E13 没有有效地节目”,
"E14 No authorization",	“E14 节目没有授权”,
"E15 Valid authorization available",	“E15 接收到授权”,
"E16 Service is currently scrambled",	“E16 当前节目已加密”,
"E17 Service is currently scrambled",	“E17 当前节目已加密”,
"E18 ",	“E18 ”,
"E19 CAT format error",	"E19 CAT 格式错误",
"E20 Not allowed in this broadcaster",	"E20 无运营商许可",

"E21 PMT format error ",	"E21 PMT 格式错误",
"E22 ",	"E22 ",
"E23 Service is currently descrambling",	"E23 当前节目正在解密",
"E24 Not allowed in this region",	"E24 无本区域许可",
"E25 Smart card not compatible",	"E25 智能卡不兼容",
"E26 Unknown service",	"E26 系统不认识当前节目",
"E27 Service is not currently running",	"E27 当前节目不在运行",
"E28 Smart card is locked",	"E28 智能卡锁定",
"E29 ",	"E29 ",
"E30 Smart card not in working period ",	"E30 智能卡不在工作时段",
"E31 Parental control locked",	"E31 父母控制锁定",
"E32 Not allowed in this country",	"E32 国家代码没有许可",
"E33 No authorization data",	"E33 没有收到授权数据",
"E34 Illegal box",	"E34 非授权机顶盒",
"E35 No signal"	"E35 没有信号",

10.3 CA Menu

10.3.1 CA Main Menu

The main menu includes basic items of TOPREAL CA, it should make up of the items as the following picture shows.

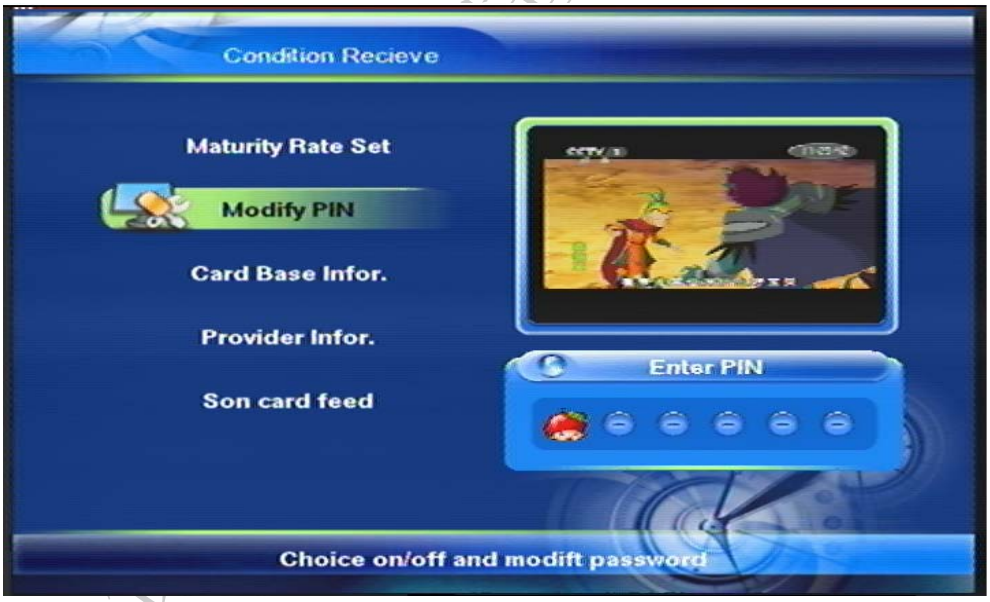


10.3.2 Maturity Rate Set

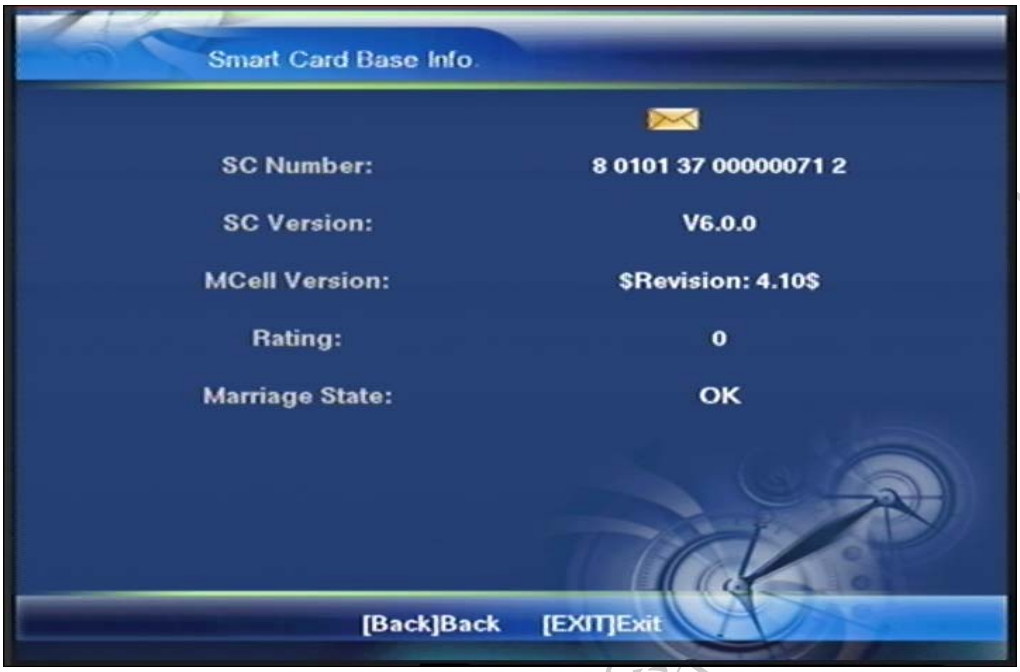
The following picture shows the menu of setting maturity rate.

Note: The range of input rate is from 0 to 10, but 0 should display as OPEN, and 1 should display as CLOSE.

10.3.3 Modify PIN



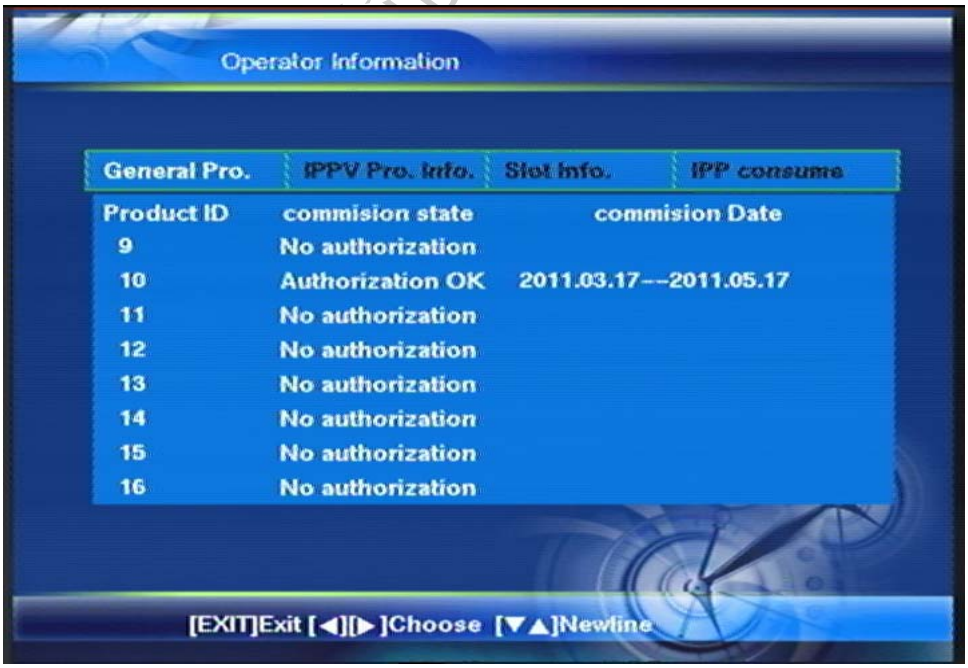
10.3.4 Card Base Info.



Note: When smart card and STB doesn't match, Marriage State should be serial number of the smart card to which STB currently matches.

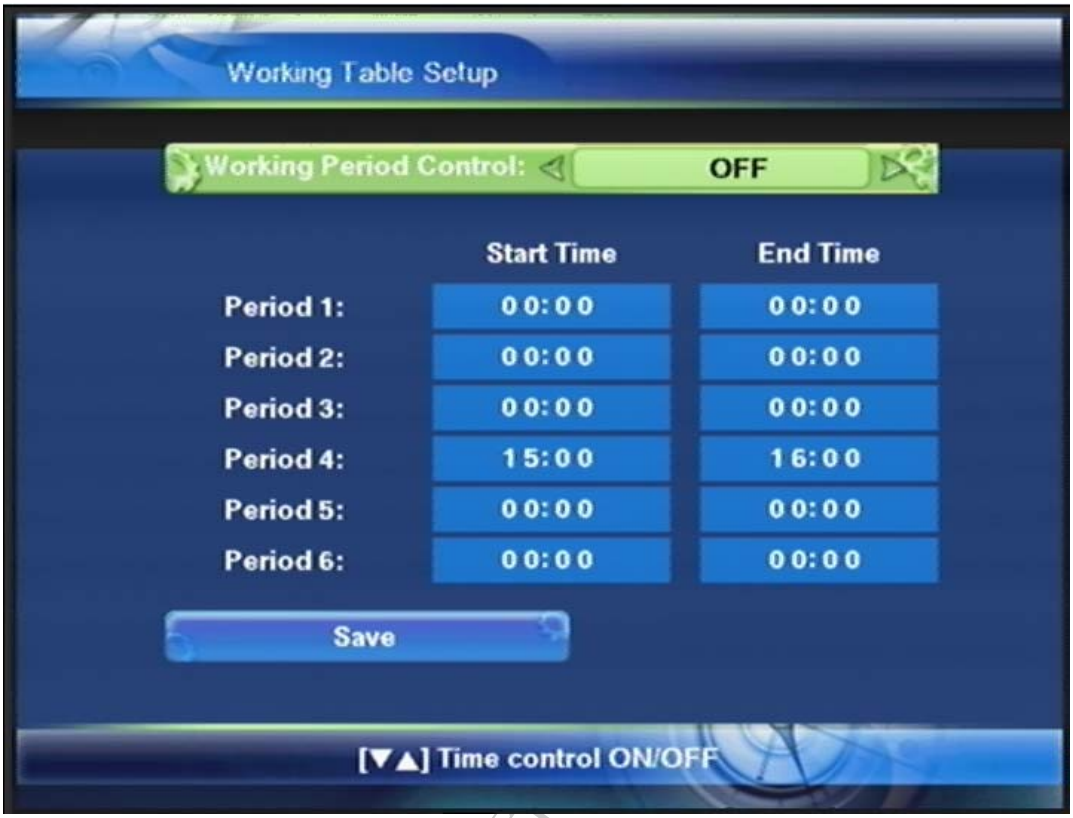
10.3.5 Provider Information

Product Authorization Information is displayed on this screen, and it contains four kinds of information. They are General Programs Authorization Information, IPPV Programs Authorization Information, Electronic virtual Money, IPP Consumption.



Notice: The product ID should be from 1 to 320.

10.3.6 Working Table Setup



This is an optional item. Whether application supports this function is determined by the broadcasting operator.

10.4 Text Message

Text messages can be divided into three groups:

1) Mail messages(mostly individual address)
2) Announcement messages(mostly broadcast)
3) Notification messages(mostly individual address)

1152 bytes (title: 128 bytes; content: 1024 bytes) can be supported by each mail or announcement and text has to wrap around. When STB receives new message, it would be stored, not necessarily in NVRAM.

This chapter, we define two types of requirements: general and enhanced requirements. General requirement is mandatory, and enhanced requirement is optional. All messages can be normal, forced or timed. General requirements are required for all manufacturers. Whether or not enhanced requirements are supported, it depends on operator. If the enhanced is not supported, Manufacturer should give an explaining to our tester, otherwise it would result in testing failure.

10.4.1 General requirements

(1) Mail message display

Normal: Store the message and display an icon clearly visible on the screen （preferable top right corner）. The user can read the message at a convenient time. The icon shall be erased as soon as the user has read the message.

Forced: the same as above

Timed: the same as normal

(2) Announcement message display

Normal: Store the message and immediately display the message including title and content on the screen.

Forced: The same as above

Timed: The same as normal

(3) Notification message display

When STB receives new notification message, Firstly it should be stored and then check remaining time of entitlement. If any entitlement is less than N days but more than 0 day, the reminder could be showed on the screen in the shape of rolling and repeats showing again and again after a period.
Note:

1) The stored space of the notification message is separated from mail and announcement, and only one notification message is existed.
2) After we reboot STB, it will check if any entitlement remaining time is less than N days but more than 0 day in the background. If it is that, the reminder is showed again and again after a period.
3) When more than one notification are received, How to process them?
   It depends on the parameter of sCreateTime in CAS_ENHANCED_MSG_STRUCT.
   The latest one should be stored and shown if necessary.

10.4.2 Enhanced requirements

(1) Mail message display

Normal: Display method is the same with general requirement.

Forced: Store the message and immediately display the message including title and content on the screen. The location, size, background, character color and the time on display are in accordance with the parameters of message data structure. An action to remove the text window is required from user.

Timed: Display method is the same as Forced and this message should be removed

from the screen automatically after a given time (see the parameter “dwRemainTime” of message data structure).

Note:

1) When the parameter “dwRemainTime’ is 0, the display method is the same as Forced.

2) Block functionality is not available for mail

3) Other unusable parameters should be ignored

(2) Announcement message display

Normal: The location, size, background, character color and the time on display are in accordance with the parameters of message data structure. The remote controller blocking function should be supported as well. If bBlockFlag is set, any button of remote control is disabled. If not, an action to remove the text window is required from user.

Forced: The same as above

Timed: The same as normal and this message should be removed from the screen automatically after a given time

Notice:

1) Block functionality is available for announcement.

2) Timed block functionality becomes invalid as soon as time is over.

10.4.3 Text Message Refreshing

Some operators want to refresh certain text messages to make sure that they are received by the subscribers. To prevent a subscriber being flood by the repetition of a same message, application shall filter out the duplicated text messages by doing the following:

1) When a text message is coming, how to judge whether it is a new message? The CRCs of message title and content should be compared with ones have already stored in message box. Notice: If it is a new, its create time should also be stored;

2) The sufficient size of the list depends on the deployment, and at least 50 items should be supported. When the list is full, the oldest message (according to the create time) must be replaced with the newly received message.

10.4.4 Test requirements

1) To ensure that all messages of stream are correctly received and the displayed content is complete, any mistake is not allowed.

2) All messages received by STB should be stored and at least 50 messages lists are

supported in message box.

3) All messages received should be displayed again on message box. Message list must contain message index, type, and title, besides, page up and page down are supported.

4) The same message is not allowed to receive again, and judgment standard is CRCs of message title and content. When you delete the message, this message needs to receive and save again.

5) If the message list is full and a new message is coming, the new message should replace the oldest one (according to the create time).

10.5 Fingerprint

Fingerprint has two display types: overt and covert. When overt fingerprint is received by STB, a twelve-character string should be displayed on TV screen and explaining text of max 158 bytes may be displayed. When covert fingerprint is coming, it is a twelve-row dot-area screen.

Fingerprint must have the highest priority over any other display, it should be impossible to block and remove the fingerprint by user any action (eg: changing channel, bringing up menu, etc).Error/status banner must be restored when the fingerprint message disappears from the screen.

Requirements of fingerprint display are given below:

10.5.1 Overt fingerprint display

Normal fingerprint: when overt fingerprint is received, the hashed serial number (just the number, without any explaining text) should be displayed on the TV screen as a banner . It shall be displayed in a random location on both X-axis and Y-axis for duration time that TRCA sends.

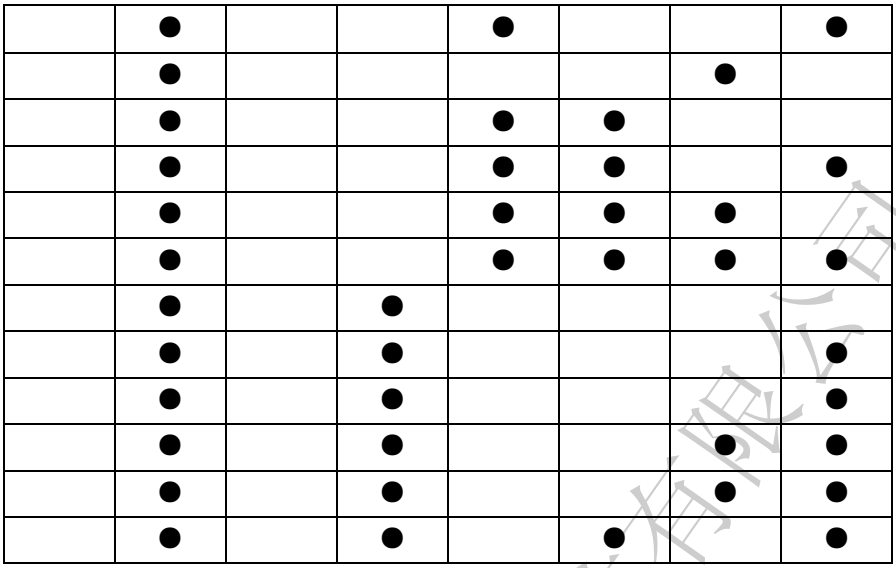
Enhanced fingerprint: If overt fingerprint is coming, fingerprint will be properly on TV screen regarding as received parameters. There are background size, background color, explaining text, character color, position, repeat time and remaining time which constructs whole fingerprint display elements.

The choice of which fingerprint to send is determined by the broadcasting operator. Both fingerprints use same data structure.

10.5.2 Covert fingerprint display

Twelve-row dot area will be displayed on screen, and every row stands for one character. In fact, every row is one byte data that means eight bits, so every row has eight dots but not every dot appears. If that bit is zero, that dot will not appear. Sketch map is

given below.



Then how to get the meaning? PLS see below:

Step 1: Convert the dot area to binary code (0: not show dot; 1: appear dot.).

Step 2: Convert each row binary code to character according to ASIC II CODE Convert Chart.

Step 3: Input the converted character string to Fingerprint decrypt window on SMS, then we can get this viewing card number (or use Fingerprint convert tool).

Row	Convert to binary code	Character
1	01001001	I
2	01000010	B
3	01001100	L
4	01001101	M
5	01001110	N
6	01001111	O
7	01010000	P
8	01010001	Q
9	01010001	Q
10	01010011	S
11	01010011	S
12	01010101	U

When you get the binary codes , you could convert them to character using ASIC II CODE Convert Chart, ASIC II CODE Convert Chart is given below:

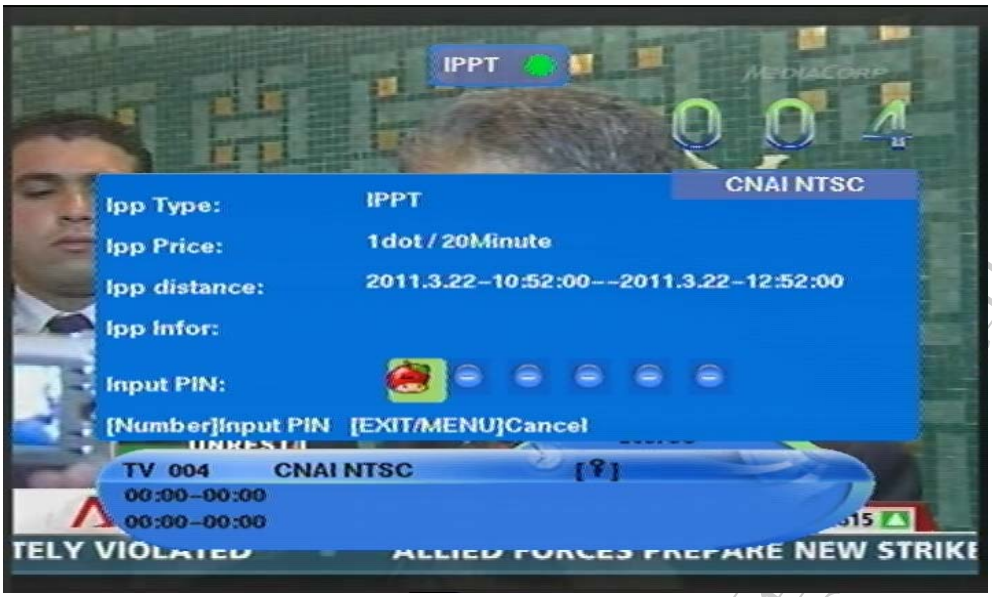
Binary	Decimal	Character	Binary	Decimal	Character
01000001	65	A	01001110	78	N
01000010	66	B	01001111	79	O
01000011	67	C	01010000	80	P
01000100	68	D	01010001	81	Q

01000101	69	E	01010010	82	R
01000110	70	F	01010011	83	S
01000111	71	G	01010100	84	T
01001000	72	H	01010101	85	U
01001001	73	I	01010110	86	V
01001010	74	J	01010111	87	W
01001011	75	K	01011000	88	X
01001100	76	L	01011001	89	Y
01001101	77	M	01011010	90	Z

We input the character string (IBLMNOPQQRRU) to Fingerprint decrypt window on SMS, then we can get this viewing card number (8010137000000910).

10.6 IPPT/IPPV

There are two types of IPP OSD: the small indicator of IPPV/IPPT and the purchasing menu of IPP. When customer switches a IPPV/IPPT service, there is a small indicator of IPPV/IPPT on the top screen and a purchasing menu, enter the smart card PIN, the program is available. The following two pictures are given as an example.



10.7 Maturity Rating Control

Here, Rating Control is a solution based on CA system, and operator can define every program rate. The viewer must press his/her PIN code in the case that the rate of this PIN code is lower than the program rate. The STB owner (e.g. parents) can also define PIN code. PLS see 10.3.2 in this chapter.

When someone switches to a parental control program, the menu of the following picture should be displayed. If right PIN is entered, the program is available.

10.8 Mother-Son Card Feed

Mother-son Card Feed is also TR CA-based solution. It is developed for preventing illegal subscriber from buying Son Smart Cards. Generally Son Smart Card is cheaper than Mother Smart Card, but provider requires that each family buys at least one STB within Mother Smart Card. In TOPREAL CAS, Son Smart Card needs feed after a period （e.g. one day）and subscriber must finish it in limited time (e.g. one minute). When an error banner of "E12 Please feed son Card" is displayed, firstly subscriber should pull out the Son Smart Card of this STB, secondly enter Mother-Son Card Feed menu on the STB within Mother Smart Card and press “OK” button and a tip of “Please insert Son Smart Card” is displayed, thirdly pull out Mother Smart Card, insert Son Smart Card into the STB, press “OK” button, if a tip of “Feed is successful” is displayed, Mother-Son Card Feed is finished.

Chapter 11 Data Structure

11.1 Introduction

This Chapter introduces the data structure used by TRCA.

11.2 Structure Descriptions

```
typedef     unsigned char   U8;          //!< 8 bit unsigned integer.
typedef     unsigned short   U16;         //!< 16 bit unsigned integer.
typedef     unsigned long   U32;         //!< 32 bit unsigned integer.
typedef     signed char      S8;          //!< 8 bit signed integer.
typedef     signed short     S16;         //!< 16 bit signed integer.
typedef     signed long      S32;         //!< 32 bit signed integer.
typedef     void             VOID;
typedef     void*            PVOID;


typedef enum
{
    TFALSE = 0,       //!< Logical false.
    TTRUE = 1,        //!< Logical true.
} TBOOL;

/* General return values */
typedef S16 TR_STATUS;

#define MC_OK                    0
#define MC_NOT_OK                1
#define MC_PARAMETER_ERROR       2
#define MC_DATA_ERROR            3
#define MC_MEMORY_RW_ERROR       4
#define MC_NOT_SUPPORTED         5
#define MC_STATE_ERROR           6
#define MC_SCSN_UNMATCHED        7
#define MC_UNKNOWN_ERROR         9    /* this is the max value. */

#define OUT    /* the parameter will be given initial data and returned data. */
#define IN     /* the parameter must be given initial data before it is called. */
#define INOUT /* the parameter must be given initial data before it is called
                 and return data during it is called. */
```

```
#define FP_SCNUM_MAX_LENGTH        22
#define FP_TEXT_MAX_LENGTH         158
#define MSG_TITLE_MAX_LENGTH       128
#define MSG_DATA_MAX_LENGTH        1024

/*DEMUX*/
#define CAS_DEMUX_INVALID_CHANNEL_ID          ((U32)0xFFFFFFFF)
#define CAS_DEMUX_INVALID_FILTER_ID           ((U32)0xFFFFFFFF)
#define CAS_DEMUX_INVALID_DESCRAMBLER_ID      ((U32)0xFFFFFFFF)

typedef enum
{
    CAS_DEMUX_DISABLE_CHANNEL,        /*stop*/
    CAS_DEMUX_ENABLE_CHANNEL,         /*start*/
    CAS_DEMUX_RESET_CHANNEL,          /*reset*/
} CAS_DEMUX_CTRL;


/*RTOS*/
union CAS_OS_Msg_Content
{
    U8      c[8];
    U16     s[4];
    U32     l[2];
    PVOID vfp[2];
};

typedef struct
{
    U32                         ulType;
    union CAS_OS_Msg_Content uInfo;
} CAS_OS_MESSAGE;


/*SMART CARD*/
typedef enum
{
    CAS_SCARD_INSERT,          /*card in*/
    CAS_SCARD_REMOVE,          /*card out*/
    CAS_SCARD_ERROR,           /*card err*/
    CAS_SCARD_UNKNOWN,         /*unknown card*/
} CAS_SCARD_STATUS;
```

```
/*CA TASK*/
typedef enum
{
    CAS_CAT_UPDATE,
    CAS_PMT_UPDATE,
} CAS_ACTION_EN;

/*Message*/
typedef enum
{
    CAS_MSG_NORMAL,
    CAS_MSG_TIMED,        /* automatic erase after date/time */
    CAS_MSG_FORCED,       /* forced display */
} CAS_MSG_CLASS;


typedef enum
{
    CAS_MSG_PRIORITY_LOW,
    CAS_MSG_PRIORITY_MED,
    CAS_MSG_PRIORITY_HIGH,
    CAS_MSG_PRIORITY_URGENT
} CAS_MSG_PRIORITY;


typedef enum
{
    CAS_MSG_MAIL,
    CAS_MSG_ANNOUNCE,
    CAS_MSG_NOTIFICATION
} CAS_MSG_TYPE;


/*CA Message Control*/
typedef enum
{
    CAS_MC_DECODER_INFO,          /* Reserved */
    CAS_MC_SERVICE_INFO,          /* 'wState' is the rating of one service */
    CAS_MC_SC_ACCESS,             /* 'wState' is the CAS_STATE_INFO index */
    CAS_MC_PIN_CODE_CHANGE,       /* Reserved */
    CAS_MC_PIN_CODE_CHECK,        /* Reserved */
    CAS_MC_MONITOR_ECM,           /* Reserved */
    CAS_MC_MONITOR_EMM,           /* Reserved */
```

```
    CAS_MC_IPP_NOTIFY,                  /* Need to process 'wState' */
    CAS_MC_IPP_INFO_UPDATE,             /* Need QueryControl */
    CAS_MC_ENHANCED_SHORT_MESSAGE, /* Need QueryControl */
    CAS_MC_ENHANCED_FINGER_PRINT,       /* Need QueryControl */
    CAS_MC_FORCE_CHANNEL,               /* Need QueryControl */
    CAS_MC_FINGER_PRINT,                /* Need QueryControl */
    CAS_MC_EMM_DOWNLOAD,                /* Reserved */
    CAS_MC_EMM_CHANGEPIN,               /* Reserved */
    CAS_MC_NOTIFY_EXPIRY_STATE, /* Reserved */
    CAS_MC_NOTIFY_CURRENT_STATE, /*'wState' is the CAS_STATE_INFO index*/
    CAS_MC_NOTIFY_SHORT_MESSAGE, /* Need QueryControl */
    CAS_MC_MAX_TYPE                     /* Reserved */
} CAS_NOTIFY_CONTROL;

typedef enum
{
    CAS_MC_QUERY_SUCCESS,
    CAS_MC_QUERY_FAIL,
    CAS_MC_QUERY_NOT_AVAILABLE,
} CAS_QUERY_STATUS;

/*CA   STATUS*/
typedef enum
{
    CAS_STATE_E00,   // Service not scrambled
    CAS_STATE_E01,   //
    CAS_STATE_E02,   // Unknown CA module
    CAS_STATE_E03,   // CA module failure
    CAS_STATE_E04,   // Please insert smart card
    CAS_STATE_E05,   // Unknown smart card
    CAS_STATE_E06,   // Smart card failure
    CAS_STATE_E07,   // Please check smart card
    CAS_STATE_E08,   //
    CAS_STATE_E09,   // Smart card EEPROM failure
    CAS_STATE_E10,   //
    CAS_STATE_E11,   // Card marriage unmatched.
    CAS_STATE_E12,   // Need to sync to master
    CAS_STATE_E13,   // No Services available
    CAS_STATE_E14,   // No authorization
    CAS_STATE_E15,   // Valid entitle availble
    CAS_STATE_E16,   // Service is currently scrambled
    CAS_STATE_E17,   // Service is currently scrambled
    CAS_STATE_E18,   //
    CAS_STATE_E19,   // CAT format error
```

```
    CAS_STATE_E20,   // Not allowed in this broadcaster
    CAS_STATE_E21,   // PMT format error
    CAS_STATE_E22,   //
    CAS_STATE_E23,   // Service is currently descrambled
    CAS_STATE_E24,   // Not allowed in this region
    CAS_STATE_E25,   // Smartcard not compatible
    CAS_STATE_E26,   // Service Unknown
    CAS_STATE_E27,   // Service is not currently running
    CAS_STATE_E28,   // Smart card is locked
    CAS_STATE_E29,   //
    CAS_STATE_E30,   // Smart card not in working period
    CAS_STATE_E31,   // Parental Control Lock
    CAS_STATE_E32,   // Not allowed in this country
    CAS_STATE_E33,   // Not receive authorization data.
    CAS_STATE_E34,   // Illegal box
    CAS_STATE_E35,   // No Signal
    CAS_STATE_E36,
} CAS_STATE_INFO;
```

/***CA DATA STRUCT***/

```
typedef struct
{
    U8 bYear[2];        // for example 1998, bYear[0]=19, bYear[1]=98
    U8 bMonth;          // 1 to 12
    U8 bDay;            // 1 to 31
    U8 bHour;           // 0 to 23
    U8 bMinute;         // 0 to 59
    U8 bSecond;         // 0 to 59
} CAS_TIMESTAMP;
```

Comments：The date format used by CAS.

/* CAS_MSG_STRUCT */

```
typedef struct
{
    U16 wTitleLen;      //Message title length
    U8  bMsgTitle[MSGTITLE_MAX_LENGTH];   //Message title
    U16 wDataLen;       //Message content length
    U8  bMsgData[MSGDATA_MAX_LENGTH];   //Message content
    U16 wIndex;         //Message index
    U8  bType;          //Message type, defined by CAS_MSG_TYPE
    U8  bClass;         //Message display type, defined by CAS_MSG_CLASS
    U8  bPriority;      //Message priority, defined by CAS_MSG_PRIORITY
    U16 wPeriod;        //Notification display interval
    CAS_TIMESTAMP sMsgTime; //No use
```

```
    CAS_TIMESTAMP sCreateTime; //Message create time
} CAS_MSG_STRUCT;
```

Comments： The message structure, it is used to store mail, announcement and notification. The parameter ”bBlockFlag” is only available for announcement Message(means block the remote control). Create time is used for replacing old message. How to judge a new message? It is depend on the CRCs of message title and content, so CRCs should be stored at the same time.

When the message type is *notification*, the ‘wPeriod’ means the display interval of the *notification*, and the time of one unit is second. Within mail and announcement, the ‘wPeriod’ doesn’t make sense. The notification is related to the *Expiry Remainder of Entitled Product*. The application of STB should check if there are some products will expire in 5 days (5 days is the default value, it can be changed by operator.). If there is any product will expire, the remainder OSD should be shown on screen. The content of OSD should be notification message and displayed periodically after one notification is received. Otherwise, the default content of OSD should be shown as “Dear Viewer, your balance is not enough. Please charge it in time.”

Notice: The days mentioned as above is a variable parameter and can be changed through the parameter “bPriority”, so when a new notification is coming, you need to update it.

```
typedef struct
{
    CAS_TIMESTAMP sCreateTime;
    U16 wIndex;
    U8   bType;                          /* CAS_MSG_TYPE */
    U8   bClass;                         /* CAS_MSG_CLASS */
    U8   bPriority;                      /* CAS_MSG_PRIORITY */
    U8   bBlockFlag;     /* Can be interrupted or not. 0: Unblock, 1: Blocking operation */
    U32 dwRemainTime;                    /* Seconds */
    U32 dwPeriod;    /* Seconds. Just for compatible with the notification of MSG1. */
    U8   bXPos;      /* 0%--100%, (0, 0) is user-defined. */
    U8   bYPos;
    U8   bWidth;     /* 0%--100%, (0, 0) is user-defined. (100, 100) is full screen. */
    U8   bHeight;
    U8   bTitleLen;
    U8   MsgTitle[MSG_TITLE_MAX_LENGTH];
    U16 wDataLen;
    U8   MsgData[MSG_DATA_MAX_LENGTH];
} CAS_ENHANCED_MSG_STRUCT;
```

Comments：

The basic requirements are the same as above (CAS_MSG_STRUCT).

/***CA Fingerprint STRUCT***/

```
typedef struct
{
```

```
    U8    bHashedNumLen; //Hashed number length
    U8 *pbHashedNum; // Hashed number
    U8    bContentLen; //No use
    U8 *pbContentData; //No use
} CAS_FINGERPRINT_STRUCT;
```

Comments： The structure is the normal fingerprint.

```
typedef struct
{
    CAS_TIMESTAMP sActivateTime;
    U32 dwOnTime;                    /* Seconds */
    U32 dwOffTime;                   /* Seconds */
    U8    bRepeatNum;                /* 0: Unique, 1---- */
    U8    bFontSize;
    U8    bXPos;          /* 0%--100%, (0, 0) is random position. */
    U8    bYPos;
    U8    bWidth;         /* 0%--100%, (0, 0) is user-defined. (100, 100) is full screen. */
    U8    bHeight;
    U8    bTextColorRed;
    U8    bTextColorGreen;
    U8    bTextColorBlue;
    U8    bTextOpacity;              /* 0%--100%, 0:transparent; 100:opaque */
    U8    bBackgColorRed;
    U8    bBackgColorGreen;
    U8    bBackgColorBlue;
    U8    bBackgOpacity;             /* 0%--100%, 0:transparent; 100:opaque */
    U8    bBlockFlag;    /* Can be interrupted or not. 0: Unblock, 1: Blocking operation */
    U8    bDispType;                 /* 0: Overt; 1: Covert */
    U8    bDispScNum;                /* If display sc number. */
    U8    bDispText;                 /* If display text message. */
    U8    bScNumLen;
    U8    ScNumString[FP_SCNUM_MAX_LENGTH];
    U8    bTextLen;
    U8    TextString[FP_TEXT_MAX_LENGTH];
} CAS_ENHANCED_FP_STRUCT;
```

Comments： The structure is the enhanced fingerprint.

/*CA Forcechannel STRUCT*/

```
typedef struct
{
    U8    bLockFlag;    //Lock flag
    U16 wNetwrokId; //Network id
    U16 wTsId;    //Transport id
    U16 wServId;    //Service id
    U16 wContentLen; //Content length
```

```
    U8 *pbContent;    //Content
} CAS_FORCECHANNEL_STRUCT;
```

Comments： The structure of forcing to change channel, it's used to store channel information which it will be changed to from stream.

```
typedef struct
 {
    U32 ulCashValue; //Cash value
    U32 ulCreditValue; //Credit value
    U16 wRecordIndex; //Last charge record index
} CAS_EPURSEINFO_STRUCT;
```

Comments： The electric purse structure, it's used to get the purse value from IC Card.

```
typedef struct {
    U8    bProductType;    // 0:IPPV; 1:IPPT
    U32 ulRunningNum; //Running number
    CAS_TIMESTAMP ulStartTime; //Start time
    CAS_TIMESTAMP ulEndTime; //End time
} CAS_IPPPRODUCT_STRUCT;
```

Comments： The IPP product structure, it's used to pass IPP product entitle information between MCELL and smart card.

```
typedef struct {
    U8    bStateFlag;          //Charging or consume flag, 0: consume; 1: charge.
    CAS_TIMESTAMP ulExchTime;    //Exchange Time
    U32 ulExchRunningNum;       //Running number
    U32 ulExchValue;         //Exchange value
    U8    bContentLen;       //Exchange content length
    U8    pbContent[34];     //Exchange content
} CAS_IPPRECORD_STRUCT;
```

Comments： The IPP exchange record structure, it's used to store IPP exchange record from IC Card, including charge or consume.

```
typedef struct {
    U16 wChannelId;     //Current channel index
    U8    bIppType;         // 0:ippv; 1:ippt
    U32 ulIppCharge;     //Charge value of one unit time
    U32 ulIppUnitTime;     //Unit time
    U32 ulIppRunningNum;     //Running number
    CAS_TIMESTAMP ulIppStart;    //Start time
    CAS_TIMESTAMP ulIppEnd;     //End time
    U8    bContentLen; //Content length
    U8 *pbContent;     //Content
```

} CAS_IPPNOTIFY_STRUCT;

Comments： The IPP notify structure, it's used to store IPP notify information from stream.

```
typedef struct {
    U16 wChannelId;    //Current channel index
    U8    bIppType;            //0:IPPV; 1:IPPT
    U16 ulPurchaseNum;    // Should be 1 always
    U32 ulIppCharge;       //charge value of one unit time
    U32 ulIppUnitTime;    //Unit time is second
    U32 ulIppRunningNum; //Running number
} CAS_IPPPURCHASE_STRUCT;
```

Comments： The IPP purchase structure, it's used to store IPP product information which will be purchased by subscriber.

typedef void (*CAS_NOTIFY)(CAS_NOTIFY_CONTROL eCtrlType, U16 wState, U32 ulParam);

Comments：

This function is called by MCELL to notify application software of the CA information.

If eCtrlType is CAS_MC_SC_ACCESS or CAS_MC_NOTIFY_CURRENT_STATE, the application should display a CA Error code (see 10.2), and wState is the index of CAS_STATE_INFO enum list.

In case of getting some kinds of CAS_NOTIFY_CONTROL(Please see the comments in enum typedef of CAS_NOTIFY_CONTROL), application software should call MC_STATE_QueryControl to get more detailed information from MCELL. The *eCtrlType* and *wState* are the input parameters while calling MC_STATE_QueryControl, and normally the *wState* is the index of message, ulParam is reserved for the future using.

While the eCtrlType is *CAS_MC_IPP_INFO_UPDATE*, the *wState* indicates whether the current channel is IPP. If *wState* is 0, it is not IPP channel. Otherwise it is IPP channel, and application should call MC_STATE_QueryControl to get IPP information by CAS_IPPNOTIFY_STRUCT and display the IPPV/IPPT indicator OSD.

While the eCtrlType is *CAS_MC_IPP_NOTIFY*, and *wState* is 0, please check if the CAS_IPPNOTIFY_STRUCT.wChannelId is same as the current playing channel. If they are same, show the IPP purchasing menu, if different, ignore the NOTIFY and clear existing menu.

While the eCtrlType is *CAS_MC_IPP_NOTIFY*, and *wState* is 1, it indicates that the current channel is IPPT and deduct from *epurse* one time automatically.