Guideline for Implementation and Use of the Common Interface for DVB Decoder Applications

DVB DOCUMENT A025
May 1997

DVB Project Office
15 may 1997

CONTENTS

1. INTRODUCTION

This document has two main purposes. The first is to explain why the Common Interface Specification [1] is designed the way it is. This will be done in the 'Rationale' sections throughout the document. The second purpose is to give guidance on how to implement and use the Common Interface. This will include recommendations for various design options where specific limits were not set in the specification.

These guidelines contain recommendations for implementation in various places which extend the Common Interface specification. These represent the best efforts of contributors to this document to ensure that modules and hosts are fully interoperable. Designers are free to accept or ignore them. However if a recommendation is ignored the designer should be confident that he fully understands the implications of doing this and the effect this may have on the interoperability of his product.

2. REFERENCES

[1] Common Interface Specification for Conditional Access and other Digital Video Broadcasting Decoder Applications, prEN50221 Draft D (23 Apr 96).

[2] PC Card Standard, Volume 2 - Electrical Specification, February 1995, Personal Computer Memory Card International Association, Sunnyvale, California.

[3] PC Card Standard, Volume 3 - Physical Specification, February 1995, Personal Computer Memory Card International Association, Sunnyvale, California.

[4] PC Card Standard, Volume 4 - Metaformat Specification, February 1995, Personal Computer Memory Card International Association, Sunnyvale, California.

3. SPECIFICATION OVERVIEW

The specification was designed in the first instance to meet Conditional Access requirements, since that was its *raison d'être*. However, almost from the start it was realised that Conditional Access was but one application that would be required by DVB hosts, and a general interface design that would allow other applications to be addressed would be much more useful than one directed just at Conditional Access. By partitioning functionality appropriately between the host and the application on the other side of the interface allowed different concerns to be neatly separated.

As an example of this, if it had been assumed that Conditional Access functionality was partitioned in some way between the host and the module-based application, as is done in most current CA system implementations, then very much work would need to be done to define a 'generic' CA function which would reside in the host, interacting with a 'specific' CA function residing in the module. Also all current CA providers, and perhaps some potential CA providers - and host manufacturers - would need to agree such a definition. This was not practical for political, technical and resource reasons, and just served to confirm the early vision of the designers that the way to go was an interface specification that was truly common and essentially free of specific application semantics.

This view also drove us to the need to transfer the whole of the MPEG Transport Stream across the interface. In principle, having agreed a Common Scrambling Algorithm, there is scope to save cost by implementing the descrambler in the host. However the descrambler is an integral component of the security of any Conditional Access system, and such security considerations and the general reluctance to agree on a common descrambler control interface made putting the descrambler on the module an essential feature of the system. Although that was the major reason for that decision it then enables very many more possibilities. The Common Interface now becomes a general-purpose MPEG input and output port, with all that entails for future functionality.

It was also not clear initially what was the best form of physical layer to adopt. PCMCIA was a very early candidate, but some work was also done to look at IEEE-1394, which was the strong favourite for Digital VCR connection, and some evaluation of the NRSS proposal from the US to use modified ISO 7816 was also done. This uncertainty confirmed another early design decision to layer the interface according to the principles

adopted in the ISO-OSI 7-layer model. Initially we thought that the Command Interface might have three or so layers. In fact it has ended up with five, one of which has two sublayers. This may seem to increase complexity, but in fact it allows the total interface functionality to be partitioned in such a way that each layer can be optimised to its task almost independently of any other layer. The approach was vindicated totally when the concept of multiple transport connections on one physical connection was introduced with virtually no design change, and the full development of the resource concept, together with the use of the session layer to make it happen, was done with no impact on the transport or any other lower layers.

4. PHYSICAL LAYER

4.1. Rationale

PCMCIA was chosen because it was suitable, relatively well specified, and was gaining rapid acceptance and deployment in the personal computer field. Initially the design conformed extremely closely to the PCMCIA Version 2.1 specification, but investigation of the implementation cost and complexity of this approach led to the current design which utilises the 'Custom Interface' provisions of the PCMCIA specification. Basic initialisation compatibility is preserved. This means that DVB-compliant hosts can accept any other PCMCIA module without damage and determine that it is not a Common Interface module. Similarly a Common Interface module can be plugged into a PCMCIA socket on any other system without damage, and the usability of it in that system can be determined.

The Transport Stream Interface is 8-bit parallel in each direction and is intended to match the type of interfaces currently being developed between front-ends and demultiplexers in MPEG-2 equipment. Using 8 bits means the data rate on each pin remains relatively modest and there is scope for a speed increase in later generations. The particular approach taken also makes for a very simple design where a host includes several module sockets and the Transport Stream Interface is daisy-chained through them.

The Command Interface has been designed to make the host side of it as simple as possible. It can be supported by simple buffering and address decoding from the host's CPU or I/O bus using programmed I/O, and for multiple-socket hosts the module bus can be connected to all sockets in parallel, with module selection done using the PC Card CE1# signal. It has also been designed to make it possible to use DMA techniques where the required data rate and module buffering capacity make this worthwhile. The buffer size negotiation process is there to allow a wide range of host and module capabilities to be accommodated, with both adjusting to the maximum common to both. There is no support for interrupts from the module to the host in the current version. It is a host responsibility to poll the interface status flags periodically to determine if any I/O operations are required. This approach has been taken to simplify both host and module interface drivers, and to enforce the convention that the host is the master in transactions with the module.

Successful communication depends on the reliability of the Command interface in this layer, since all the higher protocol layers assume that the layer immediately below is reliable. By 'reliable' is meant that data is transmitted in correct sequence with none missing and none repeated.

4.2. Transport Stream Interface

The Transport Stream Interface is 8-bit parallel in each direction, with two control lines each on input and` output and separate byte (octet) clocks for each. On each interface there is a continuous stream of bytes at the byte clock rate. This is structured into 188-byte MPEG-2 Transport Packets, and there may also be gaps in the byte stream when non-valid data is transferred. The MxSTRT signal (where x = I or O depending on direction) is valid for one byte and indicates the initial (sync) byte of each Transport Packet. MxVAL indicates valid bytes. Depending on the host implementation there are three possibilities for the operation of MxVAL:

1 Transport Packets are sent sequentially with no gaps at all, in which case MxVAL is permanently valid.

2 Transport Packets are sent as a 188-byte group with a gap of non-valid bytes between successive groups. In this case MxVAL is valid for a 188-byte period and not valid for a period before the next Transport Packet.

3 Non-valid bytes can appear within a Transport Packet as well as between Transport Packets, in which case MxVAL will go up and down as needed, but in no particularly predictable way.

Cases 1 and 2 are likely to be common in real hosts. However module designers should also allow for the possibility of case 3 unless there is general agreement amongst host manufacturers that it will not occur, either now or in future designs.

The transport stream interface allows a fairly large degree of freedom to both the host and the module designer. Depending on host and module implementations a significant PCR jitter can be created. The hosts can control the jitter produced by a module by giving it well behaved streams. The module returns a similar well behaved stream. A host sending Transport Streams with a significant amount of highly variable gaps may expect from the module a similar stream with significant PCR jitter added. The specification achieves this behaviour by indirectly limiting the PCR jitter in the returned transport stream as a function of the number of gaps present in the input transport packet.

Rule 3 under section 5.4.2 of the specification states:

"A module shall introduce a constant delay when processing an input transport packet, with a maximum delay variation (tmdv) applied to any byte given by the following formula"

For the avoidance of confusion, this means:

For the purposes of this clause, the byte positions in each MPEG transport packet shall be indexed by the term *i*. The maximum delay variation (tmdv) applied to byte *i* is given in the formula below.

$tmdv_{max} = (n * TMCLKI) + (2 * TMCLKO)$

The *delay* experienced by each $byte_i$ will depend on the input and output clock periods of the module (TMCLKI and TMCLKO), *n* and *i*. So, each byte in the packet may have a different delay. However, the *jitter* on this delay (delay variation) is independent of *i*.

4.2.1. Clock Signals

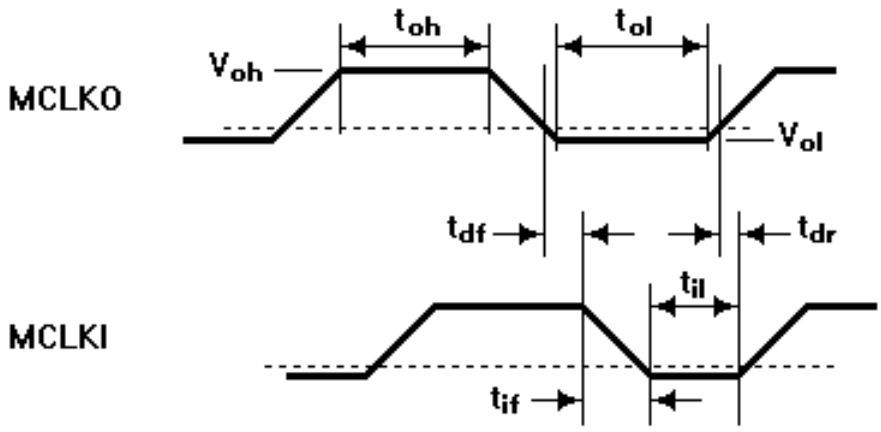
The PC Card electrical specification mainly considers memory or slow I/O cards. So, it does not consider fast clock signals such as those used by the Common Interface. This guideline addresses the specification of these signals.

4.2.1.1. Rationale

This is distilled from protracted discussion within the Common Interface Guidelines group.

For practical reasons - the use of standard HCMOS parts, and for easier EMC design - it is desirable to slow the specification of rise and fall times. Due to potential cumulative distortion, for chaining of the clocks through many cascaded modules, some form of clock regeneration is necessary in either the modules or host. The CIGG has agreed that only the module should be required to do so for economic reasons, and that the host should be permitted to implement a simple clock buffering solution, with the inevitable distortion this brings.

It has been pointed out that in these circumstances, it is very difficult to produce a specification which is not flawed when applied to the highest frequency permitted, in the sense that it will not guarantee interoperability. The argument presented at CIGG is along these lines:



The symbols used are defined as follows:

t_{ol}	output low interval on **MCLKO** (module clock output)
t_{oh}	output high interval on **MCLKO**
t_{of}	output falling edge time on **MCLKO**
t_{or}	output rising edge time on **MCLKO**
t_{df}	propagation delay of falling edge through host buffer
t_{dr}	propagation delay of rising edge through host buffer
t_{il}	output low interval on **MCLKI** (host clock output)
t_{ih}	output high interval on **MCLKI**
t_{if}	output falling edge time on **MCLKI**
t_{ir}	output rising edge time on **MCLKI**

Depending on the threshold of the sensing device in the host (and we must ignore voltage noise margins in this argument), the interval between falling and rising edges crossing the threshold can be as low as **t_{ol}** which is 40ns. The buffer might reduce this by its asymmetry **t_{df} - t_{dr}** (a falling edge may be delayed by more than a rising edge), and will then have its own rise and fall times. We can then find the guaranteed low and high intervals to the module.

The worst case low interval at the input of the second module is **t_{il} >= t_{ol} - t_{df} + t_{dr} - t_{if}** for the low interval, and **t_{ih} >= t_{oh} - t_{dr} + t_{df} - t_{ir}** for the high interval.

4.2.1.2. Guideline

The aspects addressed are:

- clock signal distortion
- clock rise and fall times
- clock signal capacitance
- over/undershoot

Clock signal distortion

The current AC specification mandates a minimum clock high and low pulse of 40ns and a period of 111ns. If a module or host uses this clock as input and to produce a buffered output, there are significant engineering challenges in ensuring that the clock is not distorted to the point where the clock specification is no longer met.

Modules shall in all cases output a clock signal MCLKO that complies with the AC specification (with the clarification that high and low intervals are measured at V_{oh}=2.4V and V_{ol}=0.5V respectively) provided that the input clock signal MCLKI supplied to the module meets the specification given below under "Clock rise and fall times". It is necessary to modify the MCLKI specification to allow practical and economic implementations of chained CI modules.

Clock rise and fall times

All statements apply both to rising and falling edges. Asymmetry in voltage levels between host and module reflects the PC-Card specification.

Module clock output MCLKO:

- V_{ol} is 0.5V and V_{oh} is 2.4V.
- Monotonic between V_{ol} and V_{oh}.
- Minimum low (at V_{ol}) and high (at V_{oh}) interval is 40ns.
- Maximum transition time between V_{ol} and V_{oh} is 20ns.
- Minimum transition time between V_{ol} and V_{oh} is 5ns (to minimise bounce and EMI).

Note that these constraints are particularly difficult to meet if the module may output a clock period of the minimum specified (111ns), as the transition time must be less (but not under 5ns for any loading condition) and the duty cycle kept very even. Modules which simply shape the incoming clock signal MCLKI to produce MCLKO are therefore considered particularly challenging and possibly impractical. To reduce the specification of the high and low intervals of MCLKO to less than 40ns would make the host buffering requirements too exacting.

Host clock output MCLKI:
- V_{ol} is 0.5V and V_{oh} is 2.8V.
- Monotonic between V_{ol} and V_{oh}.
- Minimum low (at V_{ol}) and high (at V_{oh}) interval is 20ns.
- Maximum transition time between V_{ol} and V_{oh} is 20ns.
- Minimum transition time between V_{ol} and V_{oh} is 5ns (to minimise bounce and EMI).

Note that for hosts that buffer the one module's clock to transmit to the next, there is the constraint that the sum of the asymmetry of the buffer (the absolute difference of the propagation times of rising and falling edges through the buffer) and its output transition time shall not exceed 20ns (as given by the second bullet above).

Clock signal capacitance

The following maximum capacitance specification shall be observed by all equipment:

- The maximum capacitance load presented to the module output clock MCLKO shall be 50pF and minimum 10pF.

- The maximum capacitance load presented to the host output clock MCLKI shall be 25pF, and minimum 5pF.

Under and overshoot

On all signals at all times:
$-0.5V <= V_o <= Vcc + 0.5V$

4.3. Command Interface

On the host side the interface is well specified and it is clear how data should be sent to and from the module. The specification, deliberately, says nothing about how the module should operate its side of the interface, except that whatever is done must make the host side behave in the specified manner. Thus the module side of the interface may mirror exactly the byte-oriented data transfer used by the host, or it may allow more direct addressable access to the transfer buffers. Also the module may operate in polled mode, as in the host, or it may operate in an interrupt-driven manner. These are free choices for the interface hardware designer.

All the commands on the physical interface are presumed to happen instantaneously, except for Reset (setting the RS bit in the Command register). This can take a short time to complete. The completion is signalled by the FR bit in the Status register being set.

4.3.1. Deadlock Avoidance

Module designers must be careful to avoid possible deadlocks in the operation of the Physical layer protocol. For designs which use a separate buffer for each direction of data transmission between host and module then this is not a major design problem. However low-cost module solutions may use a single buffer. In this case host and module must arbitrate to gain access to the buffer when it is empty, and to regain access once it has been used. The module designer must assure himself that his implementation is free of deadlocks when communicating with a host operating according to the specification.

Appendix B describes an example implementation, for illustration only, showing that deadlock-free implementations using a single buffer are possible.

4.3.2. Speed

The specification requires that the Command Interface Physical layer should support at least 3.5 Mbit/sec data throughput in each direction. This was a general requirement on any Physical layer chosen, and it is clear that the particular PC Card specification chosen does meet this. It does not mean that all *implementations* have to meet this requirement. The actual speed of data transfer depends on a combination of, primarily, the Physical

layer poll interval and the negotiated buffer size. Modules may only have a 16-byte buffer, and such a module would not be expected to originate or receive high data rates. Hosts, on the other hand, have to allow for at least a 256-byte buffer and must be designed to handle a commensurate data rate. A reasonable target Physical layer polling rate would be 10 milliseconds, and if each poll can handle one incoming packet plus one outgoing packet, then the effective data rate across the interface can be as high as 204.8 kbit/sec in each direction.

4.3.3. Interrupts

The original conception of the Physical layer was of a polled interface, and the specification reflects this. However it has since been pointed out that an interrupt driven interface is much more efficient, especially with multiple sockets, and the module implementation is very straightforward. It is suggested that interrupts should be supported by all modules from day 1, since this may well find its way into Version 2 of the specification as a mandatory feature of modules. The suggested implementation is as follows: DA & FR should be gated onto the IREQ# line by two new Interrupt Enable bits in the command register: DAIE (bit 7) and FRIE (bit 6) respectively. Interrupt friendly hosts would use these but polling hosts would just set the Interrupt Enable command bits to 0. The command register now becomes:

bit	7	6	5	4	3	2	1	0
	DAIE	FRIE	R	R	RS	SR	SW	HC

RS, SR, SW and HC retain their function, as described in the Common Interface specification.
When set, DAIE allows any assertion of the DA bit in the Status register also to assert IREQ#.
When set, FRIE allows any assertion of the FR bit in the Status register also to assert IREQ#.

Use is straightforward. When either DA or FR is asserted and the appropriate Interrupt Enable bit is set, then IREQ# will be asserted. An interrupt routine is called which then checks both status bits, and at this point it becomes almost exactly like the polling routine. The host must be prepared to find the FR bit reset when tested, even though an interrupt was seen, as in a single buffer design the module may have claimed the free buffer before the interrupt was serviced.

4.4. Single Socket vs. Multiple Sockets

Single socket support is straightforward, and the specification has been written to place restrictions on the module implementation to maximise host design freedom. An aim of the specification was to allow multiple socket implementations to be produced by bussing the command interface to all sockets, with only CE1# individually provided, and daisy-chaining the Transport Stream Interface with just a simple tristate buffer to bypass the data & clock signals when no module is plugged in.

4.5. DVB plus Full-Function PC Card socket

This is likely to be high cost, as the simple bus and daisy-chain approach can no longer be used. Each socket will require individual support from a control IC which integrates the PC Card standard function support with the DVB variant support.

4.6. Extenders

The specification was written with the concept of an extender in mind. This is an external device with sockets for more than one module and a plug-in lead to a socket on the host, allowing several modules to operate with a single-socket host. The design of the Physical layer allows this, and the design of the Transport layer contains features deliberately aimed at supporting this.

5. LINK LAYER

Information in this layer and all succeeding layers only refers to the Command Interface, as the layering of the Transport Stream Interface is covered elsewhere. There is, however, some discussion later of the need for many applications to extract information directly from the MPEG-2 Transport Stream.

5.1. Rationale

Only a simple protocol is provided here to perform two tasks. The first is to fragment and reassemble packets to fit the buffer size negotiated by the Physical layer on start-up. The second is to multiplex fragments from all the Transport packets currently queued to be sent through a particular socket, to fairly divide the available bandwidth between them.

5.2. Implementation Guidelines

5.2.1. Protocol

The link layer routines which operate the physical interface to send and receive fragments on the host side are relatively easy to write as that interface is well-specified. The routines on the module side will obviously be very dependent on the exact form of physical interface adopted on the module side. For designers who have to write both host and module sides of the link layer there will be benefit in making those physical layer access routines present a common interface to the link protocol routines, so that the same protocol code can be used on both sides.

In order to operate the fragment multiplexing part of the protocol each fragment contains the transport connection identifier of the transport packet from which the fragment comes. For a reason which is explained in the implementation guidelines for the Transport layer (section 5.3.2) this identifier *must* be supplied directly by the Transport layer along with the transport packet when one is passed to the Link layer for sending - the Link layer must not extract it directly from the transport packet. The Link layer should queue transport packets in such a way that independent queues for each transport connection with a send in progress are maintained, and the fragments should be picked from the packets at the head of each queue in a round-robin fashion.

One possible optimisation that the Link layer can perform is to select a buffer size to match an incoming transport packet. It can identify the first fragment of a transport packet and read the length field of the transport header, since the transport header will always be short enough to be contained in even a minimum-sized fragment.

6. TRANSPORT LAYER

6.1. Rationale

This layer provides the means whereby applications or resources on a module are connected to the host. The presence of this layer makes it easier for modules to partition their functionality, and for hosts to manage that functionality. Transport connections are *only* from module to host. The architectural model is of a single host with one or more modules connected to it. The protocol in the version 1.0 specification does not support direct transport connections between modules, nor does it support multiple hosts connected together and any need for transport connections either between hosts or between modules on different hosts. Such requirements will be addressed by other specifications or by future versions of the Common Interface specification.

However the Transport layer protocol *is* designed to support the concept of a module extender, where a single socket on a host can support several modules by means of an intermediate device which plugs into it and itself contains several sockets. The Req_T_C/New_T_C transaction allows intermediate units to set up routing tables by recognising such objects, and clearing them down again by recognising Delete_T_C/D_T_C_Reply transactions. For this reason, and because some modules may wish to utilise more than one transport connection, even hosts with only one socket must be able to support several transport connections.

The protocol continues the Master/Slave paradigm used by the Physical layer, where the host is master. The poll capability of the protocol effects this. Also there is a specific transaction to allow a module to transmit data, where the module signals that it has data to send and the host then specifically requests it. This is done so that the host can allocate a suitably sized receive buffer, which it may not otherwise do since all replies except for data are only a few bytes in size. This helps the host to optimise its buffer allocation strategy when it is handling several modules, only one or two of which are likely to be sending data at any one time. The same

consideration is not given to modules, but they have only one, or perhaps two, transport connections to service, and the size of objects sent to modules, at least in the current version, is limited practically to about 256 bytes, plus a header or two.

6.2. Implementation Guidelines

6.2.1. Minimum Implementation

On the module side the Transport layer protocol is reactive, that is, it reacts only to transport packets received from the host. Any transport packets to be sent are queued until requested by the host. The host, however, has to manage the flow of data in both directions. To request data or protocol management packets from the module it polls on all transport connections periodically. For any particular transport connection this poll must be suppressed whilst data flow transactions are proceeding, and only resumed when all data and protocol management traffic has ceased in both directions. If this were not done then it is possible, during the transfer of large transport packets to modules which only support small fragments, to get into a situation where useless poll messages begin to pile up in the host's send queue.

A potential transport protocol problem is caused by the Link layer requirement to fairly multiplex fragments from different transport connections to the same module. When a module requests a new transport connection (Req_T_C) it is important that New_T_C on the requesting transport connection should be sent *before* Create_T_C for the new transport connection. If the interface to the Link layer is done naïvely then it is possible for the link layer to send the Create_T_C before the New_T_C, since it regards them as being on different transport connections. The correct way to do it is to pass the *existing* transport connection identifier to the Link layer for both the New_T_C and the Create_T_C, and the Link layer will then sequence them correctly. Of course, the transport connection identifier *within* the Create_T_C transport packet should be for the new transport connection and not the existing one. This also illustrates the need for the transport protocol routines to use the transport connection identifier from the packet, and not the one the Link layer used for sending the fragments.

As stated in the specification the host must be able to support at least 16 simultaneous transport connections, even though it may only support one Common Interface socket. This is necessary because any module may require more than one transport connection, and also because it is possible for even a single-socket host to support multiple modules through the use of an external extender.

Both host and module must be able to do packet reassembly in the Transport layer, that is, concatenate successive T-data-more packets together with the final T-data-last packet before passing the resulting packet to the Session layer. It is not obligatory for either host or module to fragment large packets received from the session layer into multiple transport data packets. The capability was provided to support ISO 7816 protocols in the lower layers. That is not currently part of the specification but may become so in the future.

6.2.2. Multiple Connections

It is expected that modules will fall into two main categories - those that use resources and those that provide resources. However some modules may do both. In this case it is suggested that the providing and using applications should operate on different transport connections. By doing so it avoids absolutely the possibility that session packets with the same session number but for different destinations in the module will traverse the same transport connection. It is recommended, however, that a maximum of two transport connections per module is used, and that any interaction between applications on the same module, but carried out on sessions via the host, should be structured to allow this limit to be maintained.

6.2.3. Failure Modes

Both host and module should operate time-outs on requests, so that they can deal with the case of requests that are never answered. The host will need to do this for everything sent to the module, since it always expects at least a status response. The module only needs to do it for the Req_T_C request, and possibly for a Delete_T_C request.

The number of transport connections is limited, so both host and module need to be able to deal with the possibility that all transport connections are in use when another needs to be created.

There may be protocol errors, that is, the reception of an unexpected transport packet, due to a faulty implementation in host or module. Probably the easiest way to deal with this is to close down the transport connection, since protocol errors probably mean that the connection is unusable anyway.

Since this is consumer equipment, then it may be best to fail silently, though perhaps a simple display to the user that module X is faulty may help.

6.2.4. Recommended Parameter Settings

The host should apply a timeout of 300ms on all messages sent to the module. The module should apply a timeout of 300ms on Req_T_C and Delete_T_C messages.

7. SESSION LAYER

7.1. Rationale

This layer provides the mechanism by which applications gain access to the resources they want to use. The information needed to effectively use this mechanism is collected and managed by a Resource Manager in the host. The idea of a resource profile was developed very early on in discussions during development of the interface. However it was only at a late stage that this concept was fully worked out and tied together with the Session layer as its providing mechanism. The resource concept is central to the extensibility of the Application layer, and the Session layer protocol allows hosts to handle transactions to a resource provided by another module without needing to know anything about the resource other than its identifier and the transport connection over which the resource is provided.

7.2. Resources

In opening a session an application supplies a resource identifier as part of the request. To associate created sessions with resources the Session layer protocol routines on the host make reference to data structures created by the Resource Manager on the host as a consequence of operating its own Application layer protocol. The Resource Manager itself has a well-known resource identifier, so applications on modules can set up a session to this and obtain information about resources available before proceeding to use them.

7.3. Implementation Guidelines

7.3.1. Opening and Closing Sessions

Applications requesting the use of a resource use the ‘Open Session Request’ object sent to the host, to which they receive the ‘Open Session Response‘ object in reply. The host can extend the session to another module using the ‘Create Session Request’ and ‘Create Session Response’ objects. Any party to the session can close it using the ‘Close Session Request’ and ‘Close Session Response’ objects. It is important that the implementation ensures that all Request-Response transactions complete before the session carries any Application layer traffic.

All session identifiers are allocated by the host. Where sessions are to resources provided directly by the host this is a straightforward process. However, where the resource in question is provided by another module, the host extends the session through itself to the provider module. In this case two possibilities are available. The host can use the same session identifier on the two transport connections used, or it could use different session identifiers on each transport connection. The first is more straightforward to implement, but the second allows more sessions in total to be supported. In practice, with a session identifier size of 16 bits, the limit of 65535 simultaneous sessions is unlikely to be a problem and most implementors will probably choose to use a single session identifier. The implications this has for the use of transport connections has already been discussed in section 5.3.3. However the specification does allow for the alternative, and implementors should be aware that they cannot assume that a session between two modules via the host will necessarily use the same session identifier at each end.

7.3.2. Session Duration

Some resources, such as the Resource Manager, expect to support many sessions simultaneously, each of which may last for considerable periods of time. Other resources may only support one session at once, and only expect it to be active for a short period. The Session layer does not attempt to manage session duration in any way. It relies on applications being reasonable in this respect. In this context 'reasonable' means not creating a session until the resource is required and closing a session to release the resource as soon as practicable after it is no longer required.

7.3.3. Failure Modes

The Session layer response objects carry status information about the success or otherwise of a session action. Resources may be non-existent, unavailable or busy, and the Session layer will signal this. How applications

use this is up to them. In some cases it can be silently ignored. In other cases it may be necessary to inform the user of a problem and a suggested corrective action.

If a transport connection dies then all sessions carried across it are terminated by the host. Where the session is between two modules the session half on the surviving transport connection will also be killed.

7.3.4. Recommended Parameter Settings

Hosts should allow for at least 128 simultaneous sessions to be supported.

8. APPLICATION LAYER

8.1. Rationale

In order to carry out their purpose, applications running on modules make use of services provided by the host. These services are partitioned into resources, which encapsulate the total functionality available into a number of easily managed pieces. Although from an applications perspective all resources are provided by the host, the mechanisms provided enable some resources to be provided by other modules, with the host just acting as a proxy, routing resource transaction objects between provider module and user module.

As a fundamental design goal of the specification, resources are deliberately defined as simple, low-level functions. This avoids the problem of having to define any particular application so that some common feature of it can be or must be provided in the host. As an example, a Conditional Access application is responsible for the entire CA function jointly provided by host and module. The host makes available (directly or by proxy) functions such as User Interface interaction, and low-speed communications, which are generic to many potential applications. The specific support for Conditional Access is limited to a single resource providing information about services selected by the user, and the mode of interaction, if any, required with the user to make those services available for use. This is not specific to any particular CA application and indeed could be used by non-CA applications which need to be aware of the state of service selection in the host.

Both resource provider and resource user applications can be written in a modular way where interaction between resources is limited and well-understood. Each resource provides a small number of objects which define a protocol of use specific to that resource. There is no need for resources to operate within the confines of a generic Application layer protocol. Once a session is set up to a resource, the exchange of Application layer objects is defined only by the resource description in the Common Interface specification, or its annexes.

8.2. Deadlock

Whenever there is more than one application competing for resources, there is the possibility of deadlock occurring if the applications are not designed carefully. Deadlock between two applications occurs where each application is waiting for the other to release a resource before proceeding. More complicated deadlock situations can occur between multiple applications. When an application needs more than one resource to complete an operation, it should claim all resources required before starting. If, during the claim process it finds one resource busy, it should release all resources claimed thus far and then try again later. Preferably it should not wait a constant time before trying again, but wait for a randomly chosen period. This will help to ensure that, if two applications are competing for the same resources and have to back off, then they are not likely to do the same thing at exactly the same time.

9. RESOURCES

9.1. Minimum Resources

The minimum resources that any DVB-compliant host shall provide are all of those described in the main body (Section 8) of the Common Interface specification.

9.2. Response Times

These response times should take into account the maximum specified polling interval of the connection (100ms). The time should be measured in the following way :-

For the host waiting for the module :-

The time should be measured from the reception by the host of the first acknowledgement R-TPDU to the original object to the reception by the host of the first R-TPDU containing the module's response.

For the module waiting for the host :-

The time should be measured from transmission of the R-TPDU to the reception by the module of the first C-TPDU containing the host's response.

Two times are given for each resource :-

A maximum recommended response time for the resource - this is the maximum time the resource should take to respond under normal operation.

A time-out time - this is the minimum time an application should wait for a resource response before timing out and signalling an error.

9.2.1. CA Support

The module has up to 5s after reception of a CA_PMT to start descrambling and respond to the CA_PMT sent by the host object, to allow for internal CA system processing of over air messages.

However, the module should respond as fast as possible as this directly affects the speed of channel change performance.

A CA_PMT_reply object sent in response to a CA_PMT(query) object has a recommended response time of 100ms with a 500ms time-out.

If a CA_PMT object is received by a module before it has replied (if it has needed to send a CA_PMT_reply) to a previous CA_PMT then the module only has to respond to the most recent CA_PMT.

9.2.2. Host Control - Tune

No explicit response is required from the host to a Tune object, however the host is required to send a CA_PMT object when the host has tuned to the new service. The host must not send a CA_PMT for any other reason during this re-tune time to ensure the CA_PMT object is effectively the acknowledgement of the Tune object.

Recommended maximum response time = 3s.
Time-out = 10s

9.2.3. Host Control - Ask Release

The host can regain control back from the module using this object. In response the module should close the Host Control session and send a Clear Replace object if necessary.

Recommended maximum response time = 100ms.
Time-out = 500ms.

9.2.4. Date-Time Enquiry

Recommended maximum response time = 300ms.
Time-out = 1.5s.

9.2.5. Display Control

Recommended maximum response time = 300ms.
Time-out = 1.5s.

9.2.6. Keypad Control

Recommended maximum response time = 100ms.
Time-out = 500ms.

9.2.7. Comms_cmd

The host responds to comms_cmd with a comms_reply object.

The response time is measured as the time to connect to the low speed communications resource inside the host, not the time to achieve the connection at the far end of the low speed communications medium.

Recommended maximum response time = 300ms.
Time-out = 1.5s.

9.2.8. Comms_send

The host responds to comms_send with a comms_reply object.

Recommended maximum response time = 300ms.
Time-out = 1.5s.

9.3. Resource Manager

9.3.1. Purpose of this resource

This resource manages all resources known to a host on behalf of applications and other resources. It has a 'well known' resource identifier and the protocol it implements collects information about all resources and makes this known to applications and, if necessary, other resources.

9.3.2. End user of this resource

All applications and resource providers will use this resource.

9.3.3. Provider of this resource

It is provided exclusively by the host. Module-based resource providers cannot replace it by advertising newer versions.

9.3.4. Involvement of other parties

The protocol collects resource availability information from any module-based resource providers as part of its function.

9.3.5. Description of the Resource Manager

The Resource Manager collects information about all resources available to applications, either provided directly by the host or by resource providers residing on modules. It then makes this information available to applications when they first start a session to the Resource Manager. It also manages notification of changes in the availability of resources. Note that this is notification of whether the resource is available at all, not whether it is busy being used by a particular application. For completeness and regularity the list of available resources sent to an application on request should include the Resource Manager resource itself.

9.3.6. Mandatory requirements

9.3.6.1. For the host

The host must provide the Resource Manager resource.

9.3.6.2. For the module

Resource provider modules must open a session to this resource and participate in the resource information collection part of the protocol, and the resource update part.

9.3.7. Recommended Parameter Settings

The Resource Manager resource shall support at least 16 sessions.

9.4. Application Information

This resource is used by applications to present information about themselves to the host. However the more important use is for the host to be able to enter a menu tree presented by an application, if it so desires. To enter the application menu the host sends an 'enter_menu' object to the application. The application must respond by creating a MMI session to display something. In the case of an application with no menu structure this could be a simple information display which is automatically removed after a short delay or in response to a keypress. Otherwise the application can manage a single menu or a menu tree for the user to interact with the application.

9.4.1. Recommended Parameter Settings

The Application Information resource shall support at least 16 sessions.

9.5. Conditional Access Support

9.5.1. Purpose of the feature

The module has no direct interaction with the remote control of the user. Only the host has this interaction and consequently knows which programme is selected by the user. As soon as it has this information, the host must inform the module immediatly by sending the programme_number : that is the major purpose of the CA_PMT and CA_PMT_reply objects.

In order to avoid the decoding of the MPEG2 PMT Table by the module (operation already performed by the host), the host also sends to the module an abstract of this PMT containing all information required by the module to possibly descramble the selected programme.

As the user may select several programmes simultaneously, programme list management features have also been added. The host may change a whole list, or add a new programme on an existing list or update a list if a change occured in the structure of the list.

Finally, as several modules can be connected on the host, a procedure has been implemented to allow the host to know, before the descrambling starts, which module is best adapted for descrambling the selected programme. That operation is particularly complex when at least two modules are able to descramble a programme which is in Impulse Pay Per View mode. In this case, only the user can decide which module to use through an appropriate dialog.

9.5.2. End user of the feature

A CA application running in a module.

9.5.3. Provider of the feature

The CA support resource of the host.

9.5.4. Involvement of other parties

The user who selects programmes and decides between two or more module canditates through an appropriate dialog provided by the module and the host.

9.5.5. Description of the feature

Before using the feature, a session must be opened by the CA application running in the module with the CA support resource provided by the host. Its identifier is '0x00030041'.

This opening of a session is usually followed by a ca_info_enq from the host to the module and ca_info from the module to the host. This exchange is important, because it allows the host to realize a first filtering of the modules by selecting only modules that have a CA_system_id matching with the CA_system_id of the selected programme. Although this filtering is optional, we recommend host manufacturers to implement it.

The host is now ready to use the feature. There are two objects : CA_PMT from the host to the module and CA_PMT_reply from the module to the host.

9.5.5.1. CA_PMT

CA_PMT is sent by the host even when a programme in clear is selected by the user (typically a programme for which there are no CA_descriptor in the PMT). In this case, the host shall issue a CA_PMT without any CA_descriptors (i.e : CA_PMT with program_info_length == 0 and ES_info_length == 0).

CA_PMT is sent by the host to the module after one of the following events :

- a new session with the CA support resource has been opened with the CA application in the module and a programme is selected by the user.

- the user or a 'tune' command has changed the list of selected programmes or the list of selected elementary streams for an already selected programme.

- a change occured in the PMT and concerns at least one of the selected programmes. This change is signalled by a change of the version_number or a change of the current_next_indicator in the PMT table. The host does not know 'à priori' if this change affects CA operations or not. It only alerts the CA application which has the responsability to check whether the change has consequencies on the CA operation or not. Whenever it is possible, such a change should not be visible by the user (the CA application continues the descrambling while treating the object).

The host builds the CA_PMT object for one selected programme by removing all descriptors of the corresponding PMT except the CA_descriptors (which contain the PIDs of the ECMs). It is very important that the host removes all descriptors that are not CA_descriptors, otherwise the module may not reply properly. If simulcrypt techniques are in use then an entry for one programme in the PMT may contain CA descriptors for more than one CA ID. In this case the CA_PMT object will contain *all* the CA descriptors in use, that is, the receiver will not attempt to select on the basis of the CA ID(s) advertised by a particular CA application. In the case the module does not receive a comprehensible CA_PMT, it shall throw away the object. In case of repetitive errors, it may also close the CA support session and re-open it.

Upon reception of a CA_PMT, the CA application selects for descrambling only elementary streams for which there is one CA_descriptor.

The host also adds two bytes : the ca_pmt_list_management byte and the ca_pmt_cmd_id byte.

The ca_pmt_list_management allows the management of a list of selected programmes. Indeed, one CA_PMT object is sent for each selected programme. The rules for ca_pmt_list_management are the following :

- if a single programme is selected, ca_pmt_list_management == only. This CA_PMT replaces all previous programme selections.

- if more than one programme is selected, several CA_PMT are sent. The first of the list is sent with ca_pmt_list_management == first (this erases all previous programme selection). The last of the list is sent with ca_pmt_list_management == last. The others intermediate are sent with ca_pmt_list_management == more.

- if the host just wants to add a new programme, it simply sends a CA_PMT with ca_pmt_list_management == add. In this case, all previously selected programmes remain

selected. If the programme which is added already belonged to the list, then the module acts as if it received a CA_PMT with ca_pmt_list_management == update.

- if the host wants to update a CA_PMT of one of the programmes of the list it sends a CA_PMT with ca_pmt_list_management == update. This happens when the host detects that the version_number or the current_next_indicator of the PMT has changed. The CA application in the module then checks whether this change has consequences in the CA operations or not. It also happens when the list of elementary streams of a selected programme changes (e.g. : the user has selected another language). In this case, the host has to resend the whole list of elementary streams of that updated programme.

The ca_pmt_cmd_id has been created to handle properly the module selection procedure, when more than one module is potentially able to descramble the selected programme. If only one module with the correct CA_system_id is connected, the CA_PMT is sent with ca_pmt_cmd_id = ok_descrambling. In the other cases, other values are used : 'query' means that the module cannot start descrambling or mmi dialog ; 'ok_mmi' means that the module can start mmi dialog, but not descrambling ; 'not_selected' is sent after a 'query' or a 'ok_mmi' in order to tell to the CA application that another CA application has been selected. When receiving a 'not_selected' the CA application closes properly the mmi session it may have started. The description of the selection procedure is given in the attached flowcharts.

9.5.5.2. CA_PMT_reply

CA_PMT_reply is the response of the CA application running in the module. It is sent only in response of a CA_PMT with ca_pmt_cmd_id = 'query' or 'ok_mmi'. It is not sent in response of a CA_PMT with ca_pmt_cmd_id = 'ok_descrambling'. CA_PMT_reply is used during the module selection procedure. The complete description of the selection procedure is given in the attached flowcharts. CA_PMT_reply contains a parameter called CA_enable which indicates, for the whole selected programme or for each elementary_stream of the selected programme, whether descrambling is possible or not, and at which condition.

It may happen that CA_enable has different values for two ES of the same programme. See sub-clause 1.2.3 below.

9.5.6. Operational rules

9.5.6.1. General rules

- The module may not reply to a CA_PMT(programme number x, ‘query’) or CA_PMT(programme number x, ‘ok_mmi’) if it receives meanwhile another CA_PMT that deletes the programme number x from the list.

- The module must return a CA_PMT_reply with CA_enable parameters located at the same level - programme or elementary stream - as the ca_pmt_cmd_id parameters in the incoming CA_PMT.

- When receiving a CA_PMT containing several parameters ca_pmt_cmd_id equal to ‘ok_mmi’, the CA application should start as few MMI dialogs as possible (typically when it discovers that in fact all elementary streams share same ECMs and one dialogue will suffice).

- When the CA application receives a CA_PMT with a new list of programmes, it starts analysing the new list and usually stops the descrambling of the programmes of the previous list, **except** if it detects that this new list contains programme(s) or elementary stream(s) of the previous list. In this case, it should continue the descrambling of this programme or elementary stream without interruption visible by the user. For example, the only way to remove a programme from the list consists in sending a new list without the programme. In this case, the module should continue the descrambling of the other programmes of the list without interruption. Another typical example is when the host sends a new CA_PMT because the version_number of the PMT has changed. This change may not affect the CA operation and in this case, the module should not interrupt the descrambling.

- The host must send a CA_PMT even when the user selects a programme in clear (without CA_descriptors in the PMT). For example, when the user selects a scrambled programme and then a programme in clear, a

new CA_PMT without CA_descriptors is sent to the CA application in order to deselect and stop the descrambling of the previous programme.

- If the session to the CA support resource closes down, the CA application should attempt to open another session.

- CA applications currently not active for any current programmes selected by the user may create MMI sessions for user dialogue, for example to warn of an impending PPV event on another programme previously purchased by the user.

- When the PMT version changes, or a user selects a programme such that there are no CA descriptors, then the host should send a CA_PMT object containing no CA descriptors. The semantics of this empty CA_PMT is the same as for one with a ca_pmt_cm_id of 'not selected'.

9.5.6.2. Switch from scrambled to unscrambled and vice-versa

- When one programme switches from scrambled to clear, there are several possibilities:

  1. This change is not signalled in the PMT, but only in the TSC field of the packet header or in the PES_SC field of the PES header. In this case, there is no reason for the host to send a new CA_PMT to remove the programme from the list. The programme remains selected and the host keeps on sending CA_PMT when the version_number of the PMT evolves.

  2. This change results in a modification of the PMT. In this case, a CA_PMT is issued by the host.

- When one programme switches from clear to scrambled, there are several possibilities:

  1. This change is not signalled in the PMT, but only in the TSC field of the packet header or in the PES_SC field of the PES header. In this case, the host does not send a new CA_PMT. The CA application must detect that switch.

  2. This change results in a modification of the PMT (e.g : CA_descriptors are removed). In this case, a CA_PMT is issued by the host.

In both cases it is recommended that the CA application attempt to create a user dialogue to inform the user.

9.5.6.3. Host behaviour when CA_enable is not identical for all elementary streams

The CA application may reply with CA_PMT_reply containing different values of CA_enable. For example, the CA application contains the entitlements for video and audio and not for teletext. In this case, the CA application replies with ca_enable == ‘descrambling possible’ for audio and video elementary streams and ca_enable == ‘descrambling not possible’ for the teletext elementary stream. In this case, the host may select this CA application for audio and video and also select another CA application which has the entitlements for the teletext elementary stream.

Selection of CA applications for descrambling can therefore be made by the host separately for each elementary stream. Host should be able to operate in this way when it happens, but it is not mandatory. The prEN50221 indicates (see last paragraph of 8.4.3.5) that the host is not mandated to split the descrambling of several elementary streams of a same programme between several modules. When receiving a CA_PMT_reply with ca_enable values different for at least two elementary streams, such a host shall apply, for all the elementary streams of the programme, a ca_enable value equal to the lowest of the ca_enable values returned by the CA application for each elementary stream of the programme. When there are several CA applications connected to the host, the host shall immediately select a CA application answering with CA_enable == ‘01’ for all elementary streams and otherwise shall wait for all responses and select the CA application that gives the best reply, that is to say the CA application that can descramble without condition the greatest number of elementary streams. This may lead in the end to the selection of a CA application which cannot descramble all elementary streams of the selected programme. In this case, we recommend that the host displays a message saying to the user that it has selected a CA application that cannot descramble an elementary stream of the

selected programme. The host can also display a message asking the user which CA application to select when at least two CA applications are able to descramble only partly the selected programme.

9.5.7. Mandatory requirements

9.5.7.1. For the host

The host must provide the CA support resource. The process of all commands of this resource is mandatory for the host.

The module selection procedure has to be managed by the host even when only one interface is present on the host (indeed, it is possible to connect an extender on this single interface that allows to have several modules connected with the host).

The particular case of use of CA_PMT('query') described in 1.1.6 is not mandatory for the host.

9.5.7.2. For the module

The module must open a session with the CA support resource. The process of all commands of this resource is mandatory for the module.

9.5.8. Recommended Parameter Settings

The CA Support resource shall support at least 16 sessions.

9.6. DVB Host Control

9.6.1. Recommended Parameter Settings

The DVB Host Control resource shall support one session only.

9.7. Date & Time

This resource allows an application to gain knowledge of the time of day kept by the host. It also allows a simple and relatively slow alarm or time-based interrupt facility. The underlying assumption is that the host maintains its own time-of-day timer which is synchronised to the TDT table information in the current multiplex. This in itself is nominally derived from UTC with a resolution of 1 second and an undefined error, so the host's view of time is assumed to be UTC (together with a timezone offset). In practice, if the host can gain access to what it regards as a more reliable source of UTC (and timezone offset) information than TDT in the current multiplex, then it could make use of that. In this case the Date & Time resource would provide time derived from that reference.

Although the resource allows repeated date_time objects to be sent regularly to the application, it is suggested that the repeat frequency should be limited to be somewhat lower than once per second.

9.7.1. Recommended Parameter Settings

The Date & Time resource shall support at least 16 sessions.

9.8. Man-Machine Interface

9.8.1. Overview

A single MMI resource supports two styles of interaction between the module resident application and the user:

1. High level
2. Low level (with part-screen overlay or full-screen graphics)

In high level mode the application can determine the *content* of the interaction but surrenders the *method* of the interaction to the host. For example, the menu object allows the application to define a list of options. The host is required to present these to the user and communicate to the application which (if any) was selected. In many cases the output method will be characters displayed via the host's on screen display and the input method will be by pressing keys on a remote control. However, the specification does not require this method of interaction. A valid implementation could be envisaged where speech synthesis and voice recognition are used. Module applications should be designed with this wide range of implementation options in mind. In particular, carriage return and space characters may not lead to predictable results.

The low level mode uses DVB Subtitling as a method to describe bit mapped graphics when addressing the display and to place character codes or strings at specified positions on the display. This MMI method explicitly assumes that a graphic display is the output method. The input method permits a limited alphabet of virtual key codes.

For the low level methods, in contrast to the high level output methods, the host is not involved with the ergonomics of the application's display. There remain several areas for host differentiation. Some examples are given below:

- colour depth (bits per pixel)
- amount of memory provided for graphics
- drawing speed
- the number of concurrent MMI sessions supported
- the interface ergonomics for moving between applications

9.8.2. End user of the feature

An application running in a module.

9.8.3. Provider of the feature

The host.

9.8.4. Mandatory requirements

All hosts shall be able to support at least one MMI session. The session may be either high level or low level at the discretion of the application.

9.8.4.1. High level mode

When responding to enq, menu or list objects all hosts shall be able to communicate to the user at least the first 40 characters of any string and shall be able to tolerate (possibly by truncation) strings of any length. All hosts shall be able to communicate to the user menu and list objects where the number of menu/list items is in the range 0 to 20.

The choice_nb = 0xFF and item_nb= 0xFF permit the application to not know the length of the list at the time that it outputs the first MENU_more or LIST_more object. The host, therefore, is required to count the menu/list items until a "last" object is received indicating the end of the list.
All hosts shall support entry of strings of up to 40 characters in response to a enq object.

If, when responding to an enq object, the user attempts to enter more characters than the host is able to accept (or more characters than the parameter answer_text_length) then the host should give a clear indication to the user that their input is not being accepted. There is no method for the host to notify the application that user attempted to supply more input.

9.8.4.2. Low level mode

All hosts shall be able to operate in either of the two low-level modes:
- low level overlay graphics
- low level full screen graphics

It is optional for hosts to implement an object cache.

9.8.4.3. Input device

The common interface specification requires an in input method to deliver the key codes tabulated below.

key code	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
meaning	0	1	2	3	4	5	6	7	8	9	menu	ESC	⇒	⇐	⇑	⇓	BS	RC

See input device design guidelines below.

9.8.4.4. Output character codes

All hosts shall be able to communicate to the user all the displayable character codes in the ISO 6937 (1994) Latin alphabet.

9.8.5. Character Coding

Character based I/O occurs in the following cases:

- high level MMI output.
- character and string objects output delivered within DVB Subtitling data in low level graphics display modes.
- character input via the key press object.
- string input via the enq and answ objects.

9.8.5.1. Character table selection

Following the DVB SI specification, the character coding table used in strings sent to the display by the high level MMI objects and received from user input by the answ object is determined by the first bytes of each string. The default character table assumed by DVB SI and the Common Interface Specification is the ISO 6937 (1994) Latin alphabet. Where the default table is being used strings start with a character code in the range 0x20 to 0xFF (i.e. the first character code of the string). *Display support for this default table is required in all receivers supporting the Common Interface.*

The character table byte field of the display reply object lists the "string start" character table selection bytes for any character tables supported by the host in addition to the default character table. If the default table is the only one provided by the host, then the character table bytes will be a zero length list.

Character Code Table		character_table_byte(s) or string start byte(s)
ISO 8859-5 (1988)	Latin/Cyrillic alphabet	01
ISO 8859-6 (1987)	Latin/Arabic alphabet	02
ISO 8859-7 (1987)	Latin/Greek alphabet	03
ISO 8859-8 (1988)	Latin/Hebrew alphabet	04
ISO 8859-9	Latin alphabet number 5	05
Other ISO 8859		0x10nnnn
ISO 10646-1	Basic Multilingual Plane	0x11
Reserved		0x00, 0x06-0x0F, 0x12-0x1F

DVB SI provides no example of what the 0x10nnnn coding means. However, it could reasonably be interpreted as follows:

string start bytes	Character Code Table	Informative description
0x100001	ISO 8859-1	Latin 1 - covers most West European languages such as Albanian, Catalan, Danish, Dutch, English, Faeroese, Finnish, French,

		German, Galician, Irish, Icelandic, Italian, Norwegian, Portuguese, Spanish, and Swedish. Lacks some ligatures: Dutch ij, French oe and ,,German`` quotation marks. This is HTML's base charset and corresponds to the first 256 locations of the ISO 10646-1 BMP.
0x100002	ISO 8859-2	Latin 2 - covers for most Latin-written Slavic and Central European languages: Czech, German, Hungarian, Polish, Rumanian, Croatian, Slovak, Slovene.
0x100003	ISO 8859-3	Latin 3 - covers Esperanto, Galician, Maltese, and Turkish.
0x100004	ISO 8859-4	Latin 4 introduces letters for Estonian, Latvian, and Lithuanian. It is an incomplete predecessor of Latin 6.
0x100005 to 0x100009	ISO 8859-5 to ISO 8859-9	As used by DVB SI

The codes 0x80 to 0x9F and 0xE080 to 0xE09F have presentational meaning defined by DVB SI. The implementation of these presentational codes is at the discretion of the host.

Where ISO 10646-1 code tables are supported by a host, no method is currently provided to indicate the definer of codes within the private use code range (0xE000 -0xF8FF) of the ISO 10646-1 BMP. The codes 0xE080 to 0xE09F are reserved for specification by DVB SI. These should not be used by private applications. Host and module designers requiring the use of such private codes should provide a user defined object dialogue to confirm their availability.

Character codes sent to the display within DVB Subtitling data, when using the low level MMI objects, is coded using ISO 10646-1. The codes 0xE080 to 0xE09F should not be used within DVB Subtitling data (to avoid possible confusion with the DVB SI use of these codes) as character emphasis etc. is provided by other methods.

9.8.5.2. Multi byte character codes

When used, 16 bit ISO 10646-1 BMP codes shall be delivered as byte pairs, most significant byte first (i.e. the UCS-2 encoding form). Modules shall not emit data with reversed byte order (i.e. low byte first). It is *optional* for host's to implement support for reversed byte ordering (activated by reception of the illegal code 0xFFFE in place of the Byte Order Mark 0xFEFF).

When ISO 10646-1 16 bit character codes are used in high level MMI objects the string length field indicates the number of bytes in the string NOT the number of characters.

9.8.5.3. Characters via DVB Subtitling

DVB Subtitling allows character codes, or strings of character codes to be used in place of graphic object references. The character coding scheme is the ISO 10646-1 BMP. The DVB Subtitling specification does not require receivers to provide any support for this mode of operation and indicates that private agreement is required between the broadcaster ad the receiver manufacturer to ensure predictable results. In general, host and module designers requiring the use of this mode should provide a user defined object dialogue to confirm its availability.

Common interface hosts should implement output of the ISO 6937 (1994) Latin alphabet. The codes 0x20 to 0x7F map to the ISO 10646-1 codes 0x0020 to 0x007F. All hosts should implement display of the codes 0x0020 to 0x007F. No standard mapping for the codes 0x80 to 0xFF is known at the time of writing.

The Unicode consortium makes available mappings of the ISO 8859 code tables to ISO 10646-1 codes. Host implementing display of ISO 8859 code tables should use these mappings to make the characters available when addressed using ISO 10646-1 codes.

9.8.5.4. High level mode input character codes

The *answ* object returns a string of characters in response to an *enq* object. Each object can specify the character code tables used per DVB SI.
The code table used by the *answ* object shall be the same as that used by the *enq* object.

This dialogue is primarily provided to allow PIN entry. So, hosts are only required to allow input of the numbers 0-9. The minimum input character coding required is ISO 6937 (1994) (DVB SI's default character table). Thus, all hosts shall, as a minimum, support input of the characters 0-9 coded on the character codes 0x30-0x39 per ISO 6937.

It should be noted that this minimum requirement *is a subset* of that implied by the use of the DVB SI default character code table. Host support of this sub-set of the default table is implicit.

There is no method for a host to indicate that it supports input of the full ISO 6937 (1994) character table. The methods described in the specification allow the host to indicate that other complete character tables are supported for input.

The byte order for multi byte codes is described above.

9.8.5.5. Low level mode input character codes

In low level mode the host is only required to provide the key codes tabulated in the specification (and reproduced above). These don't conform to any standard, more general, character coding scheme.

9.8.6. Text direction

The sequence in which characters are presented by an application to the host is the order in which they are expected to be read by the user. So, for Latin text, the earlier characters should be the presented to the left of later characters. For Hebrew, Arabic and other right to left scripts, the reverse is true.

9.8.7. Input device design guidelines

Below is tabulated the input key code set that is supported for low level input:

key code	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
meaning	0	1	2	3	4	5	6	7	8	9	menu	ESC	⇒	⇐	⇑	⇓	BS	RC

To allow application graphics and documentation to refer to these input codes the visual representation used by the host should be understandably equivalent to the representation given in the "meaning" row of the above table. Where the host uses a different representation it's user documentation should include a description of how its representation relate to the above representations that may be expected by an application.

Certain key codes have special meaning for application low level input. These are described below. The method that generates these key codes for application low level input should be the same as that which leads to equivalent functions for the high level interface, and ideally should be the same as that used by the host's own user interfaces.

Keycode	"Meaning"	Use	Behaviour
A	Menu		TBD
B	ESC	Quit application	This key code cannot be used by applications. This key code is a message to the host that the user wishes to terminate their session with the current application. When pressed in low level mode the host should execute the following sequence of actions: • (optionally) send a key_press object carrying the ESC key code to the application. • send a close_mmi object to the application • if the application doesn't close the mmi session within a suitable time out period (such as 250 ms) the host

			should close the session using the close_session_request session protocol object.
10	BS	Cancel action	The application should abandon this user interface screen and return to the next higher level of the its menu structure. The application state prevalent before this user interface screen should be restored. The BS key should not terminate the application. A "quit application" proposal should be positively acknowledged with the RC key.
11	RC	Default action , OK , Select	This key code indicates that the user positively acknowledges the action proposed by the current user interface screen. E.g. having selected an item on a menu they confirm that they wish to act on that selection.

9.8.8. Recommended Parameter Settings

The MMI resource shall support at least 1 session. However host designers are free to implement solutions which allow for and appropriately present multiple MMI sessions to the user.

9.9. Low-Speed Communications

9.9.1. Introduction

A part of the host is an interface to a low-speed communications resource class. This will provide bi-directional communications over, for example, a telephone line or cable network return channel. This could be used to support Conditional Access functions, and be used in conjunction with interactive services.

The resource class is defined in a generic fashion so that different underlying communications technologies can be used in a common way. Resource types are defined within the class to support communication to particular types of device using the common object set. The model is a simultaneous bi-directional channel (full duplex) over which communication of arbitrary data is possible. Flow control is applied in both directions between application and host. Data is split into segments for transfer in order to limit the buffer sizes required in both application and host. The flow control protocol also limits the number of buffers required in both application and host.

9.9.2. End user of the feature

An application running in a module.

9.9.3. Provider of the feature

The host.

9.9.4. Description of the feature

Before proceeding any transaction with the low speed communication resource, a session with this resource has to be opened by the application in the module. The low speed communication identifier is '0060xxx1' where 'xxx' is the resource type. There are three types of device: modem, cable return channel, and serial port.

Four objects are defined - Comms Cmd, Comms Reply, Comms Send and Comms Rcv. Comms Cmd is sent by the application and allows several management operations on the communications resource to be carried out. Comms Reply is sent by the host and acknowledges a Comms Cmd. It also acknowledges a Comms Send, operating the outgoing flow control to the communications resource. Comms Send is a buffer of data sent to the communications resource. This is data to be sent to line. Comms Rcv is a buffer of data received from the communications resource. This is data received from the line. Buffer size and time-out values previously set by a Comms Cmd object apply to received buffers.

To use a particular communications resource, a session must be created to it using the standard session creation mechanism. Transactions using the comms objects then take place within this session.

9.9.5. Connection on a channel

The connection on a channel is only mandatory for modem and cable return channel resources. There is no reason to establish a connection when the resource is a serial port type: the application can immediately start the transaction by setting the communication parameters.

When the session is opened with a modem resource type, the application should connect the modem on a particular channel by sending the command 'Connect_on_Channel'. This requires a telephone number descriptor to be supplied. With a cable return channel, the assumption is that the host 'knows' how to connect to the cable head-end. The cable return channel descriptor allows for there to be more than one physical channel (e.g. a different upstream RF carrier) available.

The application will wait until the host replies to it with the command 'Connect_Ack'.

9.9.5.1. Connection result

If the connection descriptor is not supported by the modem, e.g. if the resource is a serial port, the host should return a 'Connect_Ack' command with return_value 'CONNECTION_NOT_SUPPORTED'.
If the host cannot establish the connection on a telephone or a cable channel, it will return the command 'Connect_Ack' with return_value 'FAIL'. When no other transaction with the modem is intended, the application should leave the session. It might open a MMI session in order to warn the user about the connection problem.
When the modem is connected, the host should return a 'Connect_Ack' command with return_value 'OK'. Then the application can set parameters for the communication session.

9.9.6. Setting parameters

The command 'Set_Params' allows to set up the buffer size and the time-out parameters. The parameter buffer_size indicates the length of the received message the host has to store, the parameter time-out indicates the maximum time the host has to wait in order to find the end of a received message.

When the application receives the command 'Set_Params_Ack' from host, it can proceed to data exchange.

9.9.7. Data exchange

9.9.7.1. Receiving data

When the resource receives data, it starts to fill its buffer 0 at first. Then when it detects the end of message: a time-out or a buffer full, it will fill the buffer 1 with the next received data.

The application can request the reception buffers via the command 'Get_Next_Buffer' with comms_phase_id '0' at first, and then alternatively 1,0,1,0 ... The host will return the command 'Get_Next_Buffer_Ack' with return_value 'OK' or 'ERROR' depending on the sequence is correct or not.

When the host has received a complete buffer and if the application has requested the corresponding phase, the host will return the message via the command 'comms_rcv' with comms_phase_id '0' at first, and then alternatively 1,0,1,0 ...

The host cannot send a buffer if the application has not requested it and the application cannot request a buffer (except at the first time) until the host has returned the corresponding buffer, if the application attempts to do so the host will return the command 'Get_Next_buffer_Ack' with return_value 'ERROR'.

9.9.7.2. Sending data

When the application wants to send data, it uses the command 'comms_send' with comms_phase_id '0' at first, and then alternatively 1,0,1,0 ... The host will return the command 'Send_Ack' with return_value 'OK' indicating that the message has been sent or 'ERROR' if the sequence is not correct.

The application cannot send a buffer (except at the first time) as long as the host has not acknowledged the previous one, if the application attempts to do so the host will return 'Send_Ack' with return_value 'ERROR'.

9.9.8. Disconnection on a channel

When the application wants to terminate transactions with a modem, it sends the command 'Disconnect_on_Channel'.

If the connection is not supported by the resource, for instance a serial port, the host will return the command 'Disconnect_Ack' with return_value 'CONNECTION_NOT_SUPPORTED'.

When the modem is disconnected, the host will reply 'Disconnect_Ack' with return_value 'OK'.

9.9.9. Close session

When the application does not need the modem anymore, it should close the session with it.

9.9.10.Example of a low speed communication transaction

application	commands	host
a session with a modem type resource is requested	(...open session...) ↔	if there a free resource, a session is granted
request for connection on a channel	comms_cmd (Connect_on_Channel) →	
	comms_reply (Connect_Ack) ←	the modem is successfully connected
the communication parameters are passed to the host	comms_cmd (Set_Params) →	
	comms_reply (Set_Params_Ack) ←	the communication parameters are set up
A message has to be sent: using comms_send_more/last commands	comms_send (phase 0) →	
the length is longer than the buffer size	comms_send (phase 1) →	
	comms_reply (Send_Buffer_Ack, phase 0) ←	an ack. is returned, when the first part of message has been sent
	comms_reply (Send_Buffer_Ack, phase 1) ←	an ack. is returned, when the last part of message has been sent
a message has to be received	comms_cmd (Get_Next_Buffer, phase 0) →	
	comms_reply (Get_Next_Buffer_Ack, phase 0) ←	the request is acknowledged
	comms_rcv (phase 0) ←	a message has been received in buffer 0: using the commands comms_rcv_more/last
next part of message to be received	comms_cmd (Get_Next_Buffer, phase 1) →	
	comms_reply (Get_Next_Buffer_Ack, phase 1) ←	an ack. is returned
	comms_rcv (phase 1) ←	a message has been received in buffer 1
data exchange terminated: the modem has to be disconnected	comms_cmd (Disconnect_on_Channel) →	
	comms_reply (Disconnect_Ack) ←	the modem is disconnected
The session is closed	(...close session...) ↔	The resource is released

9.9.11. Mandatory requirements

9.9.11.1. For a telephone modem type resource

All commands are supported by the resource. The connection_descriptor_type has the value 01. If another value is received, the host will return the error 'CONNECTION_NOT_SUPPORTED'.

9.9.11.2. For a cable return channel resource

All commands are supported by the resource. The connection_descriptor_type has the value 02. If another value is received, the host will return the error 'CONNECTION_NOT_SUPPORTED'.

9.9.11.3. For a serial port type resource

All commands are supported by the resource. If the resource receives 'Connect_on_Channel' or 'Disconnect_on_Channel', the host will return the error 'CONNECTION_NOT_SUPPORTED'.

9.9.12. Recommended Parameter Settings

Multiple instances of Low-Speed Communications resources are represented by different resource types, and one instance may also be represented by multiple resource types to indicate the various speed options available. However, each resource instance will only support one session so.

9.10. Authentication

This optional resource is defined to facilitate a host-to-module authentication process should this be desired by particular host/module combinations. Where module support for authentication is not expected in a particular case then the host must allow unauthenticated transactions between itself and the module. This authentication process can only be used to deny service to a module where the module purports to be of a type that is expected to support the defined authentication process for that type.

9.10.1. Recommended Parameter Settings

The Authentication resource shall support at least one session.

9.11. EBU Teletext Display

This optional resource is provided to allow application access to the teletext display system where one is provided on the host. It is an alternative screen display mechanism to the MMI resource and may be appropriate for certain applications.

9.11.1. Recommended Parameter Settings

The EBU teletext resource will most probably only support one session.

9.12. Smart Card Reader

This optional resource allows application access to a smart card reader on the host, or on another module This will not in general be the smart card reader used by a conditional access card, if such is used in the system, but instead it will be a card reader made available for other card-based transactions, such as bank card or electronic cash card operations.

9.12.1. Recommended Parameter Settings

The protocol allows for more than one session to be active to the resource at any one time. However only one of these sessions can be 'connected' to the card reader at any one time. The non-connected sessions can only read reader and card status information. The Smart Card reader resource should allow up to 16 sessions to each card reader resource instance.

9.13. DVB EPG Future Event Support

This optional resource is provided to allow an Electronic Programme Guide to obtain entitlement status information from a CA application about future events, if such information is available. The protocol also includes facilities to allow conditional access MMI sessions to take place to give entitlement if required, such as the purchase of Pay-per-View events. Note that the resource is provided by the EPG, and it is a CA appication decision whether to create a session to it to provide the required information.

9.13.1. Recommended Parameter Settings

The DVB EPG Future Event Support resource should support at least 16 sessions.

10. ERROR MANAGEMENT

10.1. Introduction

Errors in Application Layer are not specified, it is supposed that both host and module are complying with that specification. But in order to make a proper and reliable software, the design could not assume that the environment is perfect, and processes should survive in case of error intrusion from faulty implementations.
This document is intended to define some rules of implementation in order to improve interoperability between different implementors.

This document does not cover errors relevant to lower layers, it is assumed that Apdus are delivered correctly to Application Layer by Session Layer (when session with resource has been granted).

This document is intended to be discussed by the CIGG members.

10.2. Application Layer Failures

10.2.1. Protocol errors

A protocol error may be caused by a command which cannot be proceeded because the process is not in an adequate state: eg. a modem is asked by an application to send data but communication parameters have not been set up before or the connection with a server has not been established.

10.2.2. Command errors

A command error may be caused by the reception of uncorrect Apdus: Application Object Tag not supported, data length not supported, data field not understood.

10.2.3. Transport disconnection

The transport disconnection (due to the physical removal of a module or the logical deletion of the transport connection) may have a consequence on Application Layer: eg. a resource should be released when the applicative owner is disconnected, otherwise the resource becomes unavailable forever.

10.3. Implementation guide-lines

10.3.1. Transport disconnection

The transport disconnection will have an impact on Session Layer, Resource Manager, Application Information, CA support implementations.

If the transport connection has been created for an application, all resources currently used by it (if any) should be released.

- *Rule for the host Session Layer:*

The host will send a Close_Session_Request to all remote resources. The resource provider (host or module) will be responsible to release properly the resource(s): eg. to disconnect the low speed communication or to close gracefully the MMI dialogue.

- *Rule for the Application Information and CA support resources:*

They should remove the disconnected CA application from their own application list.

If the transport connection has been created for a resource provider, all opened sessions (if any) should be closed.

- *Rule for the host Session Layer:*

The host will send a Close_Session_Request to all applications. Any tentative to make use of the disconnected resources will be ignored by the host.

- *Rule for the Resource Manager:*

The Resource Manager should update its resource list and if this results in any change it should send Profile_Change to all applications.

10.3.2. Command errors

- *Generic rule:*

If a resource or an application receives a command with an Application Object Tag not specified or not relevant to itself, it should ignore the command and do nothing.

10.3.2.1. Resource Manager

Profile Enquiry
Any command with length size not null is ignored

Profile Changed
Any command with length size not null is ignored

Profile Reply
Length and data are depending on the number of resources supported by the Resource Manager. It seems reasonable that the Resource Manager may support up to 64 resources. The minimum set of resources that the host shall provide is: Resource Manager, Application Information, CA Support, Host Control, Date-Time, MMI, Low Speed Comm.

- *Rules for an application:*

Any command with length size not equal to N * 4, N in the range 7 to 64, is erroneous and is ignored.
Any resource identifier not supported by application is ignored.

- *Rules for the Resource Manager:*

Any command with length size not equal to N * 4, N in the range 0 to 57, is erroneous and is ignored.
Any command with resource_identifier() not supported is ignored, ie. any one from this list: Resource Manager, Application Information, CA Support, Host Control, MMI which shall be provided by host only.
Any command with a list containing identical resource_identifier() is ignored: a resource_identifier() should be unique.

10.3.2.2. Application Info

Application Info Enquiry
Any command with length size not null is ignored

Application Info

The menu_string_length indicates the number of characters in the menu_string, the maximum length allowed is 40. So the length size can have the maximum value 46. A null menu_string_length is allowed, in this case the host will be able to implement a default menu_string.
Any command with length size out of the range 6 to 46 inclusive is ignored.
Any command with application_type not specified is ignored.
Any command with text_char not specified is ignored.

Enter Menu
Any command with length size not null is ignored

10.3.2.3. CA Support

CA Info Enquiry
Any command with length size not null is ignored

CA Info
It seems reasonable that the CA application may support at least one CA_system_id and a maximum of 16 CA_system_id.
The host shall support any type of CA_system_id (it shall not interpret it).
Any command with length size out of the range 2 to 32 inclusive is ignored.

CA PMT
ISO/IEC 13818-1 specifies that the length of TS_program_map_section shall not exceed 1024 bytes. It is reasonable to apply this limit to the CA_PMT since only CA descriptors are transmitted.
Any command with length size exceeding 1018 bytes is ignored.
Since a CA_descriptor may have a maximum length of 256 bytes, any command with program_info_length or ES_info_length exceeding 257 is ignored.
Any command with ca_pmt_list_management not specified is ignored.
Any command with ca_pmt_cmd_id not specified is ignored.
Any command with CA_descriptor() not supported at program or ES level is ignored (eg. CA_system_id not supported).
Any command with elementary_PID not existing or not relevant is ignored.
All reserved fields should be set to ‘0’.

CA PMT Reply
Any command with program_number not relevant is ignored.
Any command with version_number / current_next_indicator not relevant is ignored.
Any command with CA_enable not relevant is ignored.
Any command with elementary_PID not existing or not relevant is ignored.
All reserved fields should be set to ‘0’.

10.3.2.4. Host Control

Tune
Any command with length size not equal to 8 is ignored.
Any command with network_id not existing is ignored.
Any command with original_network_id not existing is ignored.
Any command with transport_stream_id not existing is ignored.
Any command with service_id not existing is ignored.

Replace
Any command with length size not equal to 5 is ignored.
Any command with replaced_PID not existing or not_relevant (eg. reserved PIDs) is ignored.
Any command with replacement_PID not existing or not_relevant (eg. reserved PIDs) or not compatible (eg. audio replaced by video) is ignored.

The maximum number of replacement_ref per application is 256 (0 to 255). The maximum number of replaced PIDs within the same replacement_ref is infinite: a more realistic number might be 16.

Clear Replace
Any command with length size not equal to 1 is ignored.
Any command with replaced_ref not existing or not granted is ignored.

Ask Release
Any command with length size not null is ignored.

10.3.2.5. Date and Time

Date Time Enquiry
Any command with length size not equal to 1 is ignored.

Date Time
Any command with length size not equal to 5 or 7 is ignored.
Any command with UTC_time not specified is ignored.

10.3.2.6. MMI

Close MMI
Any command with length size not equal to 2 is ignored.
Any command with close_mmi_cmd_id not specified is ignored.

Display Control
Any command with length size not equal to 1 or 2 is ignored.
This command is an exception of the generic rule: there is a response or an acknowledge transmitted in the Display_Reply command.
The response to any command with display_control_cmd not specified will be Display_Reply with display_reply_id equal to “unknown_display_control_cmd”.
The response to any command with MMI_mode not specified will be Display_Reply with display_reply_id equal to “unknown_mmi_mode”.

Display Reply
Any command with display_reply_id not specified is ignored.

Keypad Control
Any command with length size out of range 1 to 19 is ignored.
Any command with keypad_control_cmd not specified is ignored.
Any command with key_code not specified is ignored.

Keypress
Any command with length size not equal to 1 is ignored.
Any command with key_code not specified is ignored.

Text
Any command with length size exceeding 40 is ignored.
Any command with text_char not specified is ignored.

Enq
Any command with length size out of range 2 to 42 is ignored.
Any command with text_char not specified is ignored.

Answ
Any command with length size out of range 1 to 41 is ignored.

Any command with answ_id not specified is ignored.
Any command with text_char not specified is ignored.

Menu
Any command with erroneous TEXT() is ignored.
Any command with choice_nb not equal to 255 and out of range 0 to 10 is ignored.
Any command including more than 25 commands TEXT() is ignored.

Menu Answ
Any command with length size not equal to 1 is ignored.
Any command with choice_ref not relevant is ignored.

List
Any command with erroneous TEXT() is ignored.
Any command with choice_nb not equal to 255 and out of range 0 to 10 is ignored.
Any command including more than 25 commands TEXT() is ignored.

10.3.2.7. Low Speed Comm

Comms Cmd
Any command with comms_command_id not specified is ignored.
Any command with erroneous connection_descriptor() is ignored.
Any command with buffer_size exceeding 254 is ignored.
Any command with comms_phase_id not equal to 0 or 1 is ignored.

connection_descriptor
Any connection_descriptor with connection_descriptor_type not specified is an error.
Any connection_descriptor with telephone_descriptor not supported is an error.
Any connection_descriptor with channel_id not supported is an error.

Comms Reply
Any command with length size not equal to 2 is ignored.
Any command with comms_reply_id not specified is ignored.
Any command with return_value not specified or not relevant with the comms_reply_id value is ignored.

Comms Send
Any command with length size exceeding 255 is ignored.
Any command with comms_phase_id not specified is ignored.

Comms Recv
Any command with length size exceeding 255 is ignored.
Any command with comms_phase_id not specified is ignored.

10.3.3. Behaviour in case of errors

10.3.3.1. Resource behaviour

- *Generic Rule:*

Any command not expected is ignored (in case of Protocol error). However the resource may decide to close down the session with the faulty application in some cases like risk of deadlock, too many errors or timeout errors.

- *Rules for the Application Information resource:*

The generic rule is applied. The host may discard the faulty CA application and may signal the error to the user.

- *Rules for the CA support resource:*

The generic rule is applied. The host may discard the faulty CA application and may signal the error to the user. If a MMI dialogue is in process, the host may close it.

10.3.3.1. Application behaviour

- *Generic Rule:*

Any command not expected is ignored (in case of Protocol error). However the application may decide to close down the session with the faulty resource in some cases like risk of deadlock, too many errors or timeout errors.

11. GENERAL IMPLEMENTATION GUIDELINES

11.1. Initialisation

Two scenarios are considered. In the first a module is plugged into a host already switched on. In the second a host containing two or more modules is switched on.

When a module is plugged in the pin configuration is such that the power and ground are applied first, then the data & control signals, and lastly the module detect pins. The module executes a power-on reset which, amongst other things, places the output signal pins in a high impedance state so that no signals can interfere with operations on existing modules. When the host detects the presence of the module it reads the Card Information Structure in the module‘s Attribute Memory and determines from this the operating voltage (in conjunction with the status of the Voltage Sense pins) and whether in fact this is a DVB Common Interface module. If it is not then the initialisation process stops here. If it is then the bypass path for the Transport Stream Interface in the host is switched off and the appropriate value is written into the Card Configuration Register to configure the module into its operating mode. At this point the MPEG-2 Transport Stream is flowing through the module but not, as yet, being descrambled. There is an unavoidable introduction of several bytes of false data into the Transport Stream at this point due to the added delay of the module. This completes the PC Card initialisation process, much more detail of which is given in the PC Card Standard. Whilst this has been going on the module, having executed a power-on reset, waits for the PC Card configuration to complete. During this time it performs any other configuration required to prepare itself for the signal to move to its operating mode.

The host now initialises the Physical layer of the Command Interface by performing the buffer size negotiation process. No explicit initialisation of the Link layer is required. The host now proceeds to create a Transport layer connection to the module. Once this has been established the module requests a session to be opened to the host‘s Resource Manager, which has a “well-known” Resource ID, and this will invariably be granted. The Resource Manager now sends a Profile Enquiry to the module to discover if the module is offering any resources. The module replies with a Profile Reply which will contain a resource list if the module offers any. Otherwise the list is empty. The Resource Manager now sends the module a Profile Change, in response to which in most cases the module sends a Profile Enquiry. The host then replies with a Profile Reply listing all the resources that are available for the module to use. If the module were purely a resource provider, then it may not perform this last step, as it has no need to use resources (other than the Resource Manager) made available by the host. If, at the previous stage, the module had supplied a list of resources it offered, then the host would update its master resource list and, if this list had changed, it would also send out Profile Change objects on all other active Transport Connections. At this point the module is ready to perform its tasks.

Application modules will create a session to the Application Information resource to pass application information and to manage application menu entry points. Once the session is created the host sends an Application Info Enquiry to the application, which responds with Application Info.

A Conditional Access application will also then create a session to the CA Support resource to allow CA information from the SI and information about user-selected services to be given to the application. Once the session is created the host sends a CA Info Enquiry to the application, which responds with CA Info. The host may then enter into a subsequent dialogue with the CA application to determine which selected services the CA application can descramble and under what conditions.

At this point the initialisation process is complete.

In the case where a host already containing two modules is switched on, the process is rather similar. The host will go through its own initialisation process. At the same time the modules will do a power-on reset and wait. Depending on the host implementation, it will perform the PC Card initialisation process either sequentially on the modules or in parallel. The upper-layer initialisation already described could also be done sequentially or in parallel. However one useful optimisation would be to perform the Resource Manager protocol for all modules together. In this case the host performs Profile Enquiries for all opened Resource Manager sessions to gather all possible resource information before issuing any Profile Change objects. This could be done by judicious use of time-outs and contextual information in the host about the number of modules (and transport connections) present.

11.2. Disconnection

When a module is unplugged the module sense pins will disconnect first. There is then a small amount of time before the data and power pins disconnect for the host to tidy up. The main function to be performed is the reinstatement of the bypass path in the host for the MPEG-2 Transport Stream. This will cause a disturbance to the data flow by the sudden removal of the delay through the module, but this is unavoidable.

11.3. Hot Swapping

For design purposes one must assume that users will plug and unplug modules at any time during operation of the host. Therefore both host and module must be designed to allow this. In general PCs deal with this by isolating the PC Card socket through interface devices when no card is plugged in, only powering up when a card insertion is detected. However, for low-cost host design it is convenient to operate the command interface part of the Common Interface as a bus. In this case some of the data and control pins on the PC Card socket remain ‘live’ all the time and host and module must be designed to minimise the chance of mis-operation of the bus (glitches) when a module is inserted.

The PC Card socket is designed with different pin lengths for different signals. The power & ground pins are longest (5.00 mm), the signal lines are next (4.25 mm), and the Card Detect pins are shortest (3.50 mm). Assuming reasonably conservative values for insertion velocity, the signal pins will begin to connect between 3 and 5 ms after the power pins connect, and the Card Detect pins 3-5 ms after that.

There are two possible glitch sources when a card is inserted. The first is such a low impedance on the module power lines that it causes a brief voltage reduction on the receiver power supply. This can be mitigated by limiting the power line capacitance in the module, providing a reasonable power line reservoir capacitance on the power pins in the host to source the current surge, and providing a small but non-negligible impedance (R and/or L) in the host between this reservoir capacitance and other circuits in the receiver. The values of C, L and R should be designed appropriately to give a module power risetime within, say 3 ms. At the same time the module should be designed to place all its signal lines in a high-impedance state within the 3 ms limit.

The second possibility is for the card insertion to cause glitches on the signal pins. This should not be from resistive loading, but will arise from capacitive loading by the module - the signal line in the host will source a charge current pulse for the node capacitance in the module. The goal is to maximise the time period for the node charging process, thus minimising the peak charge current. This leads to design techniques such as minimising the main bus impedance in the host, but increasing appropriately the bus impedance on the feeds to each socket. This would normally be by providing wider PCB tracks on the main bus and thinner ones on the side branches. This adds both series L and R. Once the module has powered up the signal lines will stay in a high-impedance state until the host detects the card insertion and begins to perform read/write operations to it.

On withdrawing a module the host has 3 ms or more to close down the connection to that module neatly. This will involve closing sessions and transport connections. There is less opportunity for the module to detect that it is going to be unplugged, however it may be possible to use the Card Detect pins without going outside the specification for the operation of these signals.

11.4. Protocol Layer Implementation

In a layered specification such as this it is useful to have a basic design strategy which can be applied to all or most layers. As packets of data flow back and forth between module and host there is a lot of passing of data and control transactions between layers, and a regular pattern for this makes the writing of each layer much easier, as many of the design concepts used to write one will also be applicable to the others. The following design discussion suggests one or two possibilities for doing this, but it is by no means exhaustive and software designers who have done this sort of thing before in other ways may well wish to capitalise on that experience instead.

The functional interactions between a layer and the one below it can probably be resolved into four basic ones: Send to the layer below; Receive from the layer below; perform an action on the layer below; be notified of an event in the layer below. These allow data to be passed back and forth between layers, connections to be created and deleted, and the appearance and disappearance of connections to be notified. By limiting the interaction between layers in this way, it also makes it less likely that unforeseen and unwanted interactions between layers are designed into the software.

Communication between layers can be simply by function call, though this means that a chain of function calls right up or down the layers builds up. This is not in itself a problem unless there is limited memory available for the stack frame, but it then becomes important to make sure that everything gets sequenced properly. For example, if a session is created to a resource and then a response is sent back to the creating application, it is important to make sure that a side effect of the ‘create’ operation on the resource is not to start up the Application layer protocol for that resource. This can only happen *after* the response has gone back. The proper sequencing is to create the management data that will be used by the session layer for that session, then to return the response to the requesting application, and only then to signal the session creation to the resource.

An alternative means of communicating between layers is to use data and command queues (it might be a shared queue) between layers. In this case it is easy to ensure that all the necessary work is completed in one layer before control is relinquished. However the implementation is more complex because all control has to return to some kind of scheduler which decides to which layer control should be given based upon what is in the various inter-layer queues.

In the Application layer itself a regular interface between all resources and the Session layer is important. By associating the ‘receive’ function for each resource with the management data structure used by the Session layer, for example by using function pointers, a very efficient message dispatch can be carried out, since there is not the overhead of a switch or case statement, or even worse, a chain of ‘if’ statements.

12. ENVIRONMENTAL

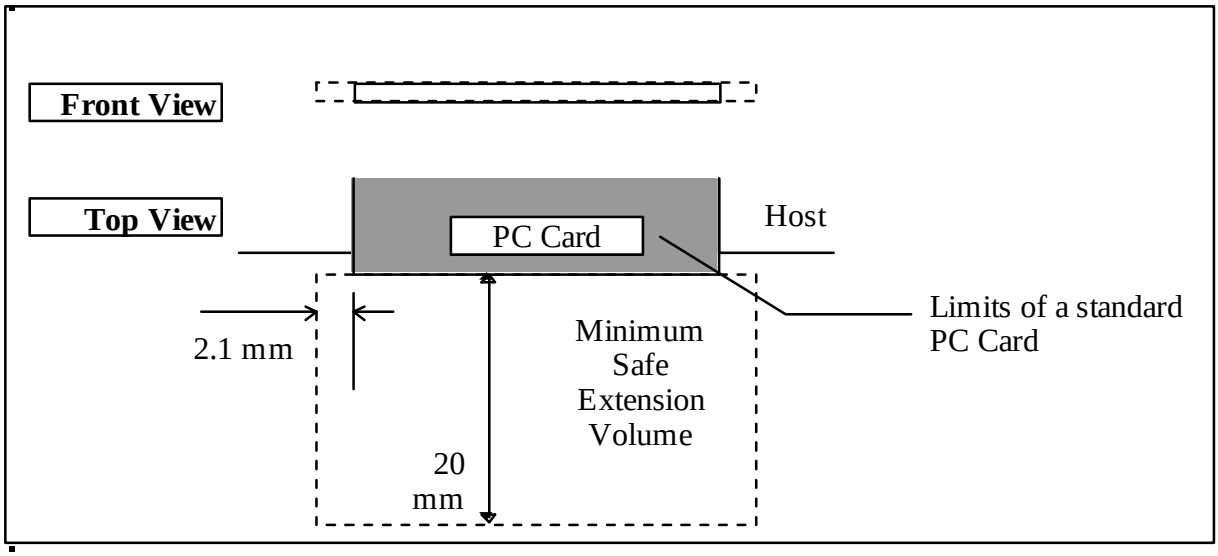
12.1. Mechanical Guidelines

The standard states that hosts shall accept both Type I and Type II PC cards. Support for Type III cards is optional.

In addition to the above, host designers should make provision for extensions, outside of the host, beyond the standard PC Card mechanical format. This allows the PC Card itself to be extended beyond the standard PC Card mechanical format. It also makes provision for cables to be connected to the card.

The extension volume is constrained in the following way:

- Shall not be thicker than the PC Card’s T dimension.
- May, beyond the PC Card’s depth L, extend up to 2.1 mm either side of the card’s W dimension.
- May, beyond the PC Card’s depth L, extend up to a further 20 mm.
- Provision should be made to allow a cable, of potentially infinite length, to exit from the PC Card.



12.2. Module environmental considerations

The DVB Common Interface Specification requires detachable modules to operate at environmental temperatures up to 55º C without any substantial degradation in reliability over the design lifetime of the module. The specification restricts the maximum rise above ambient to 15º C within the region in the host where the module is situated while the module is dissipating full rated power. It assumes both the host and module have been powered long enough for both to reach a stable operating temperature. This restriction is necessary to accommodate the maximum temperature limitation of lithium batteries which may be present in detachable modules. Such limitations are usually in the range of 55º -- 60º C.

The specification recognises that the 55º module environment temperature may be exceeded in a relatively small population of hosts where the ambient temperature may occasionally be higher than 40º C. Those modules will experience reduced battery life depending on the length of time during which the temperature exceeds the limit recommended by the battery manufacturer. The extent by which battery life is thereby reduced is described in derating curves supplied by battery manufacturers. However, given the variable ambient conditions in which consumer electronic devices are used, it is difficult to accurately predict the time intervals during which modules' temperature may rise above their 55º nominal maximum.

It is incumbent upon both host and module manufacturers to take reasonable and appropriate steps to assure proper temperature control within the module operating environment.

12.3. Host device manufacturer guidelines

Host device manufacturers must assume that battery-equipped modules will be used in their equipment and must take steps to assure that the operating environment conforms to the specification. This implies selecting a location for the module socket which is most favourable to maintaining module environment temperature within limits that will not adversely impact battery life. This should include performing a thermal analysis to determine the best module location and ventilation measures followed by confirmation testing of the design using module(s) that consume the maximum specified power. Additional issues that need to be considered include module orientation and module-to-module proximity in multi-module hosts. In general, heat may be more effectively dissipated using vertical orientation, and modules must be separated sufficiently to avoid trapping heat between them.

12.4. Module manufacturer guidelines

Module manufacturers must select battery technologies that are best able to withstand the higher operating temperatures likely to be encountered in the host device in certain ambient environments. Batteries should be selected, in part, on the basis of derating curves that are most favourable to such temperature extremes. Manufacturers should place batteries in the location recommended in Volume 3, Figure 3 (*Physical Specifications*) of the PCMCIA specification. They should also locate heat-producing devices in an area of the

PC Card as far removed as possible from the standard battery location. Moreover, they should realise that host manufacturers may have provided special ventilation for the area which the PC Card specification recommends for battery placement. Generally, this is the area within approximately 2.0 cm from the end of the card opposite the connector.
Module manufacturers should minimise the constant drain on batteries to that necessary to achieve the operational requirements of the module. Designs are preferred which consume battery power only when the module is detached from the host, as battery life can approach shelf life, notwithstanding the possibility of occasional elevated temperatures.

12.5. Module Grounding

12.5.1. Rationale

EMC is a particularly difficult area to manage, especially in a scenario such as with the common interface where EMC responsibility is difficult to partition between the module and the host. There is no simple solution to the problem, as can be seen from existing markets (PC-Card for example) where there is no arbitration process for this partitioning.

The route taken here is to strongly advise the use of ground clips as described in the PC-Card specification, and recommend sensible engineering practices.

12.5.2. Guideline

The following design principles represent good engineering practice.. The details given below may only be relaxed when there is sufficient confidence with respect to the EMC performance of the resulting implementation. .

On the host, both ground clips shall be provided. -A ground plane parallel to the module within 5 mm of the module may be provided to realise additional EMI shielding, Such a ground plane shall be connected directly to the ground clips. Similarly, the ground clips shall be directly connected to a metal chassis plate through which the module may protrude if such a metal chassis plate is provided. This metal plate could form part of an enclosing EMI shield. Selection of ground clip geometry shall take into account the possibility of a Smart Card in the module as given below.

All four ground pins from the module connector should have a minimum impedance path to the electrical ground of the host. A way to achieve this is to solder the pins directly (maximum wire length 10 mm) to the ground plane of the PCB with the tracks carrying the signals to and from the module. In any case the routing of these signal tracks should ensure that any signals originated by the host meeting any impedance in the module, and signals within specification originating in the module driving into the host (including possible feedthrough and reflections on inputs and power pins) will not be radiated above the permitted EMI levels. It is advisable to apply transmission line and antenna theory when laying out signal tracks to anticipate EMI levels.

On the module, two ground clip points shall be provided as specified in the PC-Card standard, except as provided in the next paragraph. Each of these will be connected to either one or two metalised covers on the module (it is permissible to have the two covers isolated and each separately connected to one ground clip, or only one metalised cover connected to both clips). By "metalised cover" is meant a conductive metal coated plastic or pressed metal cover extending over one entire face of the module. In most cases it is beneficial to prevent any low impedance path from these ground clips to the signal ground of the module, thus preventing ground loop problems and antennas. All four signal ground pins should be directly connected via a low impedance path to the electrical ground of the host. A way to achieve this is to use minimal length tracks to a ground plane which must cover the maximum practical amount of the PCB.

Modules that incorporate a Smart Card do not have to provide the ground clip for the entire 3.3 mm depth of the guide rail. On either one of the edges of the rail, up to 1.1 mm need not be available for contact to allow for an inserted Smart Card. Host ground clip designs must take this into account, including the possibility that there may be an unoccupied slot in the guide rail (Smart Card removed).

For ESD protection, the module must be able to tolerate an ESD discharge from any of its power pins during insertion and from the casing to the internal board, without damage. The host must also be able to tolerate discharges to the power pins.

13. CONSTRAINTS AND INTERPRETATION

13.1. Bit and Byte Order Interpretation

Both the Transport Stream Interface and the Command Interface are byte-oriented, and the most significant bit is defined as D7 in both cases. Thus the bit order within bytes is defined. The byte order is also defined such that the more significant bytes of a multi-byte quantity are the earlier ones to be transferred across the interface. This is the so-called “Big-endian” convention. The operation of interface hardware invariably operates so that arriving bytes of a byte stream are placed at successively higher addresses in a memory buffer. Thus the most significant byte of a quantity is at the lowest addressed byte of that quantity. Processors that operate this convention are also called Big-endian, and examples include the Motorola 68000 family and the Sun Sparc family. No ordering conversions in software are therefore required for these processors. However, some processor families, notably the Intel 80x86 series, observe the Little-endian convention where the most significant end of a multi-byte quantity is at the highest addressed byte. Software for such processors will require byte order conversion routines in order to operate correctly.

Note that the Card Information Structure read when configuring the PCMCIA interface uses the Little-endian convention, that is, more significant bytes of multi-byte quantities in the Attribute memory of the card are at higher byte addresses. Beware of this change of convention between PCMCIA data and data in all layers above.

In practice, another common characteristic of processors, that of address alignment restrictions, means that both categories of processor will probably require combined extraction, placement and conversion routines. For example, when a received message is being analysed and it contains a 32-bit uimsbf quantity, the address of that quantity in the message buffer may be such that the processor would generate a bus alignment error if a direct 32-bit access was attempted. A suitable extract routine will pick the quantity byte-by-byte and place it into a properly aligned variable in the correct order for use by the following application code. Similarly a suitable placement routine will place a multi-byte quantity at an arbitrary address in a buffer and in the correct order when constructing messages. Such routines will be an indispensable part of any software implementation.

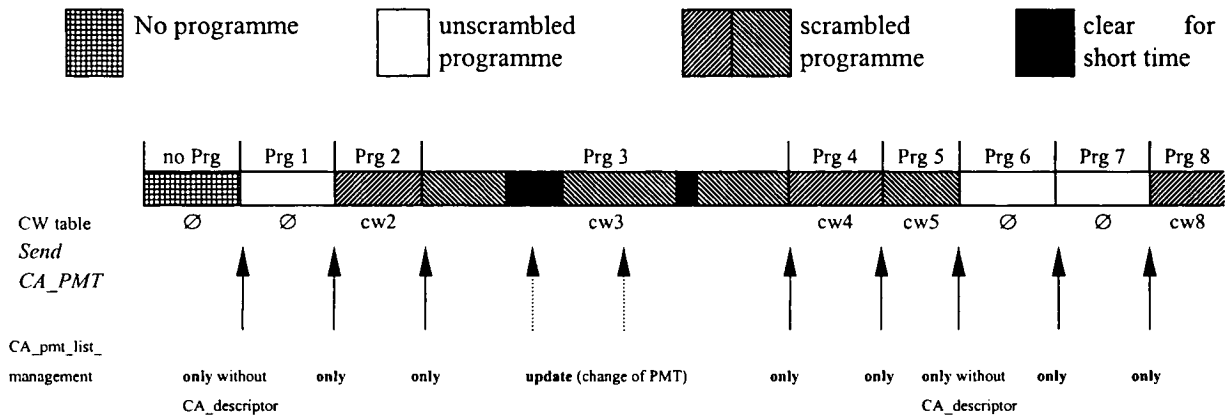
APPENDIX A: DIAGRAMS AND FLOWCHARTS OF CA_PMT OPERATION

A.1. Diagrams describing the use of the ca_pmt_list_management parameter

The following diagrams give some example of use of the ca_pmt_list_management parameter. Each arrow corresponds to the sending of a CA_PMT parameter. The 'CW table' line indicates which CWs are stored by the CA_application (e.g.: cw1 means that the CA application keeps in memory the control word(s) of programme 1).

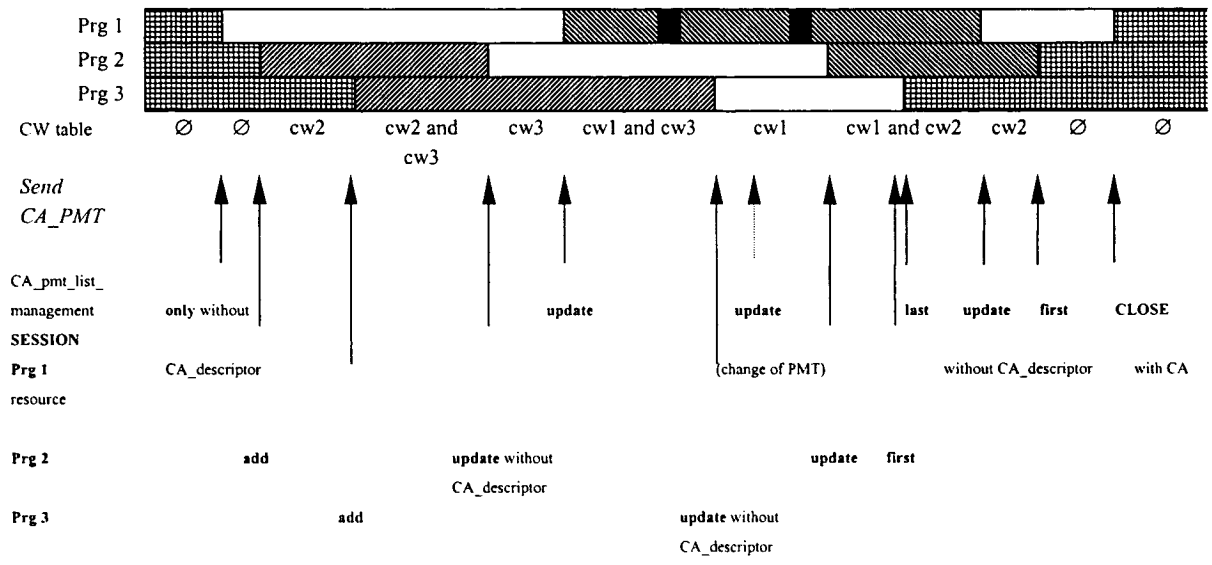
Example of CA_PMT management for one selected programme.

In this example, only one programme at a time is selected by the user. The user switches from one programme to another.



Example of CA_PMT management for three selected programmes

In this example, the user selects several programmes (up to three) in parallel and then deselects them. At the end, when no programme is selected, the host sends a CLOSE_SESSION object to close the session with the CA application resource.

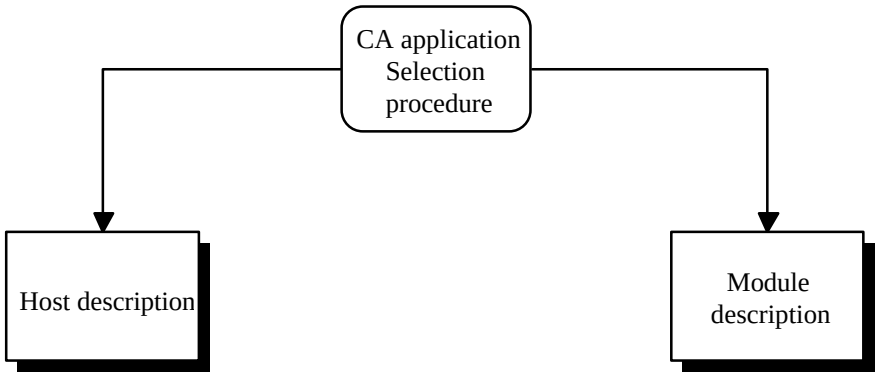


A.2. Flowcharts describing the use of CA_PMT and CA_PMT_reply

The following flowcharts describe the host and module use of CA_PMT and CA_PMT_reply in the case of :

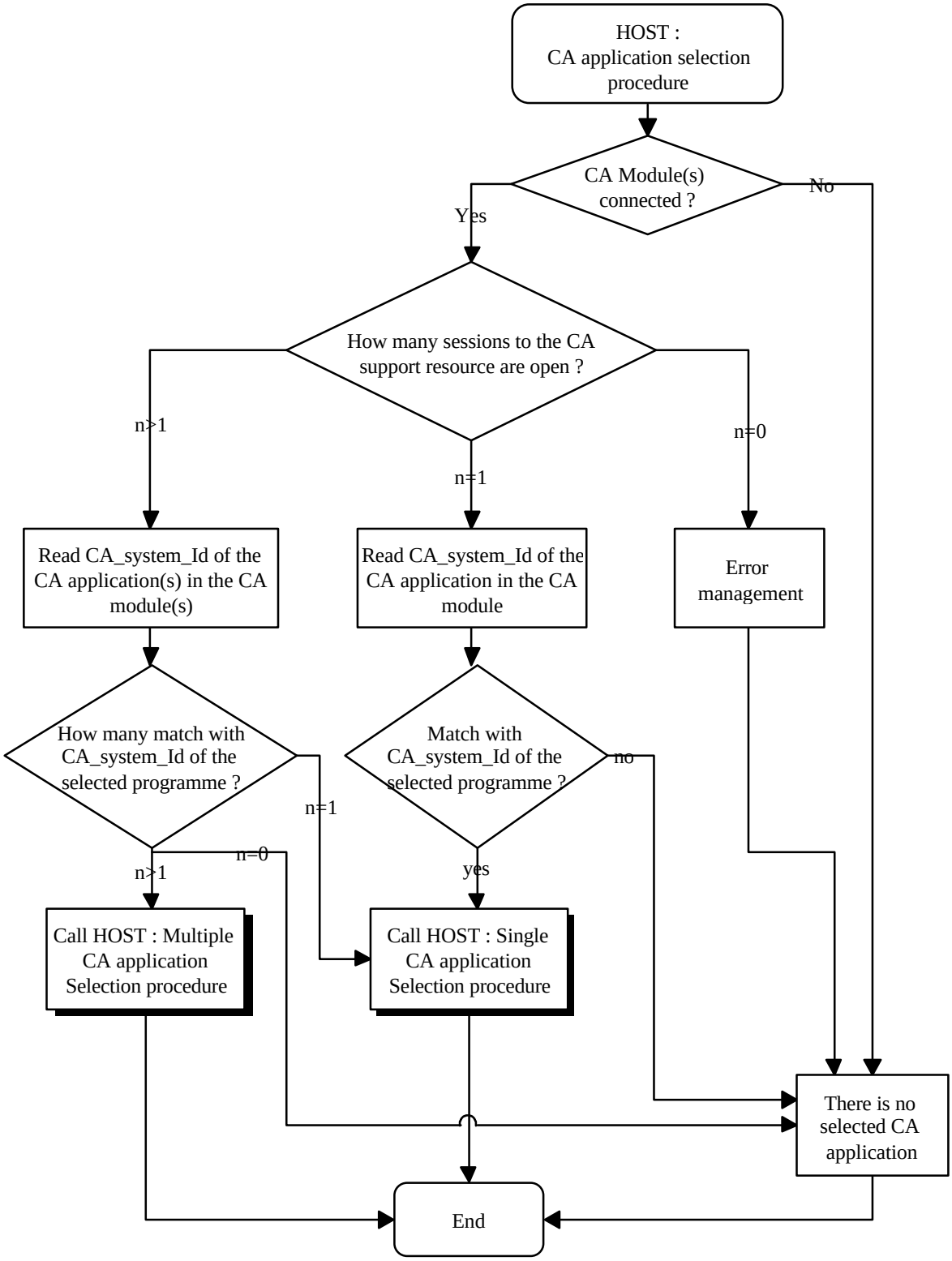
1. a unique programme selection (the ca_pmt_list_management is not described).

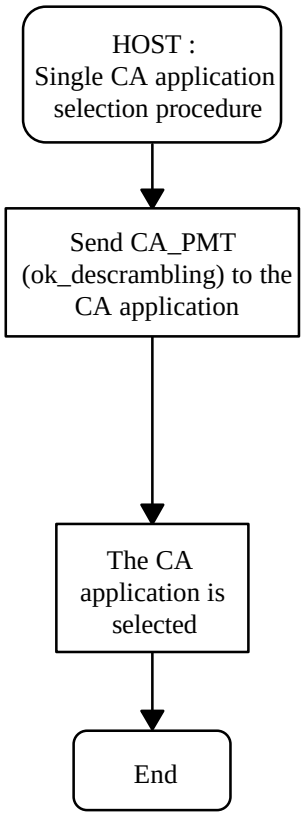
2. same ca_enable applies for all elementary streams of a selected programme (the splitting of the descrambling between several CA modules is not described, although this can be easily derived from the flowcharts when considering that the CA application selection which is described in the flowchart for a whole programme can be performed by the module separately for each elementary stream).



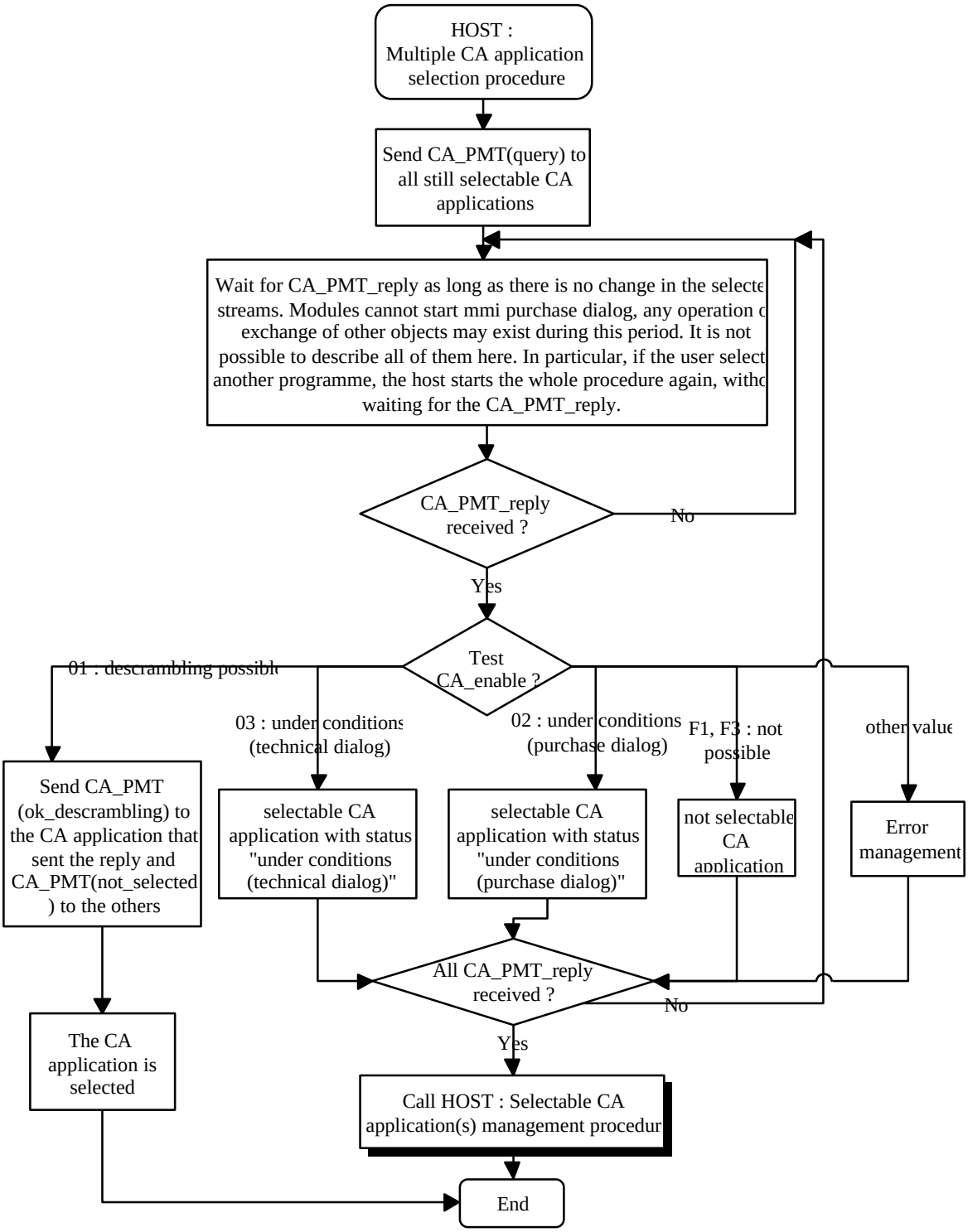
Important remark : at any time, the HOST process described in the following flowcharts can be interrupted by an external event (e.g : the user selects another programme)

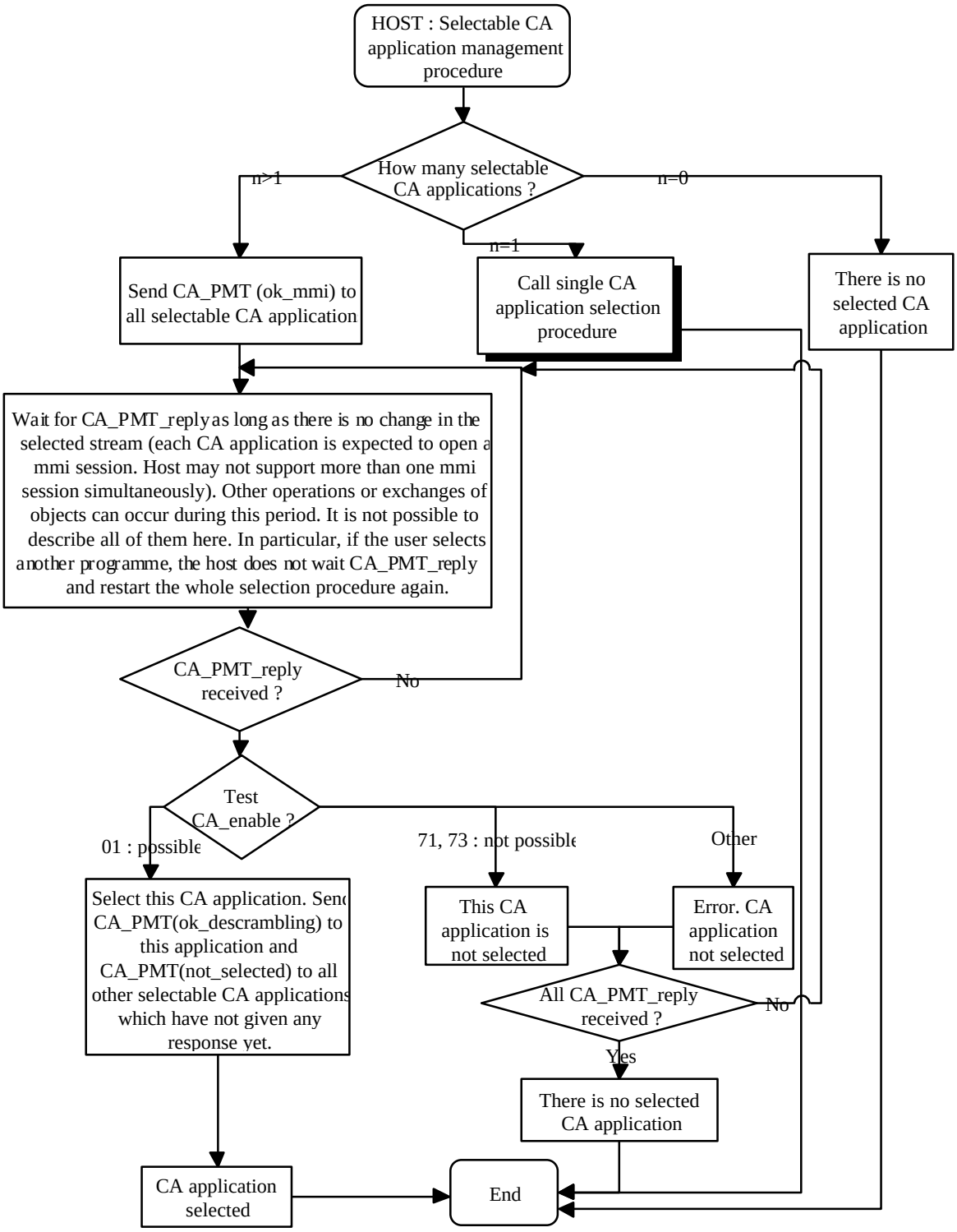
Important remark : at any time, the Module process described in the following flowcharts can be interrupted by an external event (e.g : the user selects another programme)

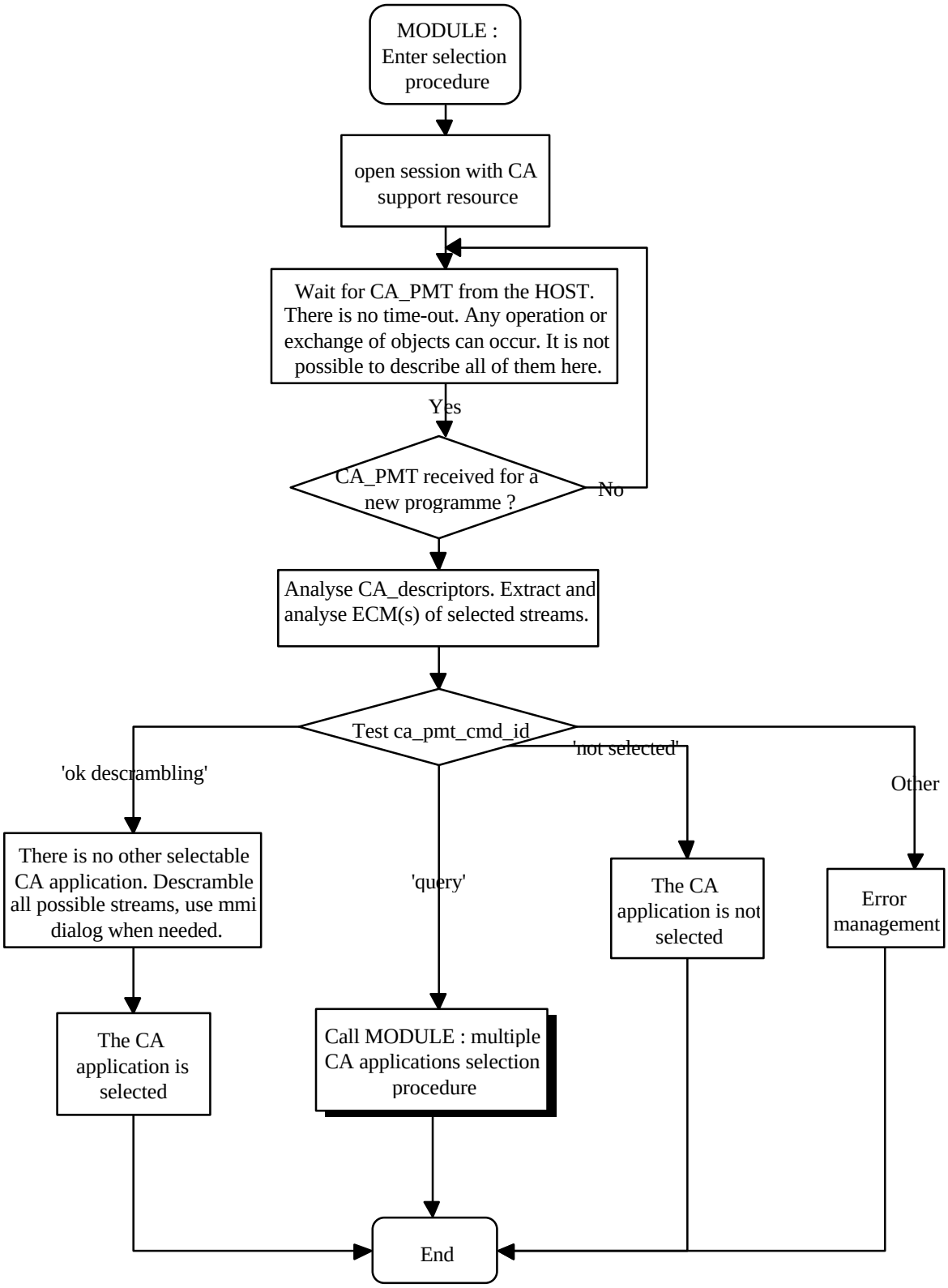

HOST :
CA application selection procedure
CA Module(s) connected ?
No
Yes
How many sessions to the CA support resource are open ?
n>1
n=1
n=0
Read CA_system_Id of the CA application(s) in the CA module(s)
Read CA_system_Id of the CA application in the CA module
Error management
How many match with CA_system_Id of the selected programme ?
Match with CA_system_Id of the selected programme ?
no
n=1
n=0
n>1
yes
Call HOST : Multiple CA application Selection procedure
Call HOST : Single CA application Selection procedure
There is no selected CA application
End

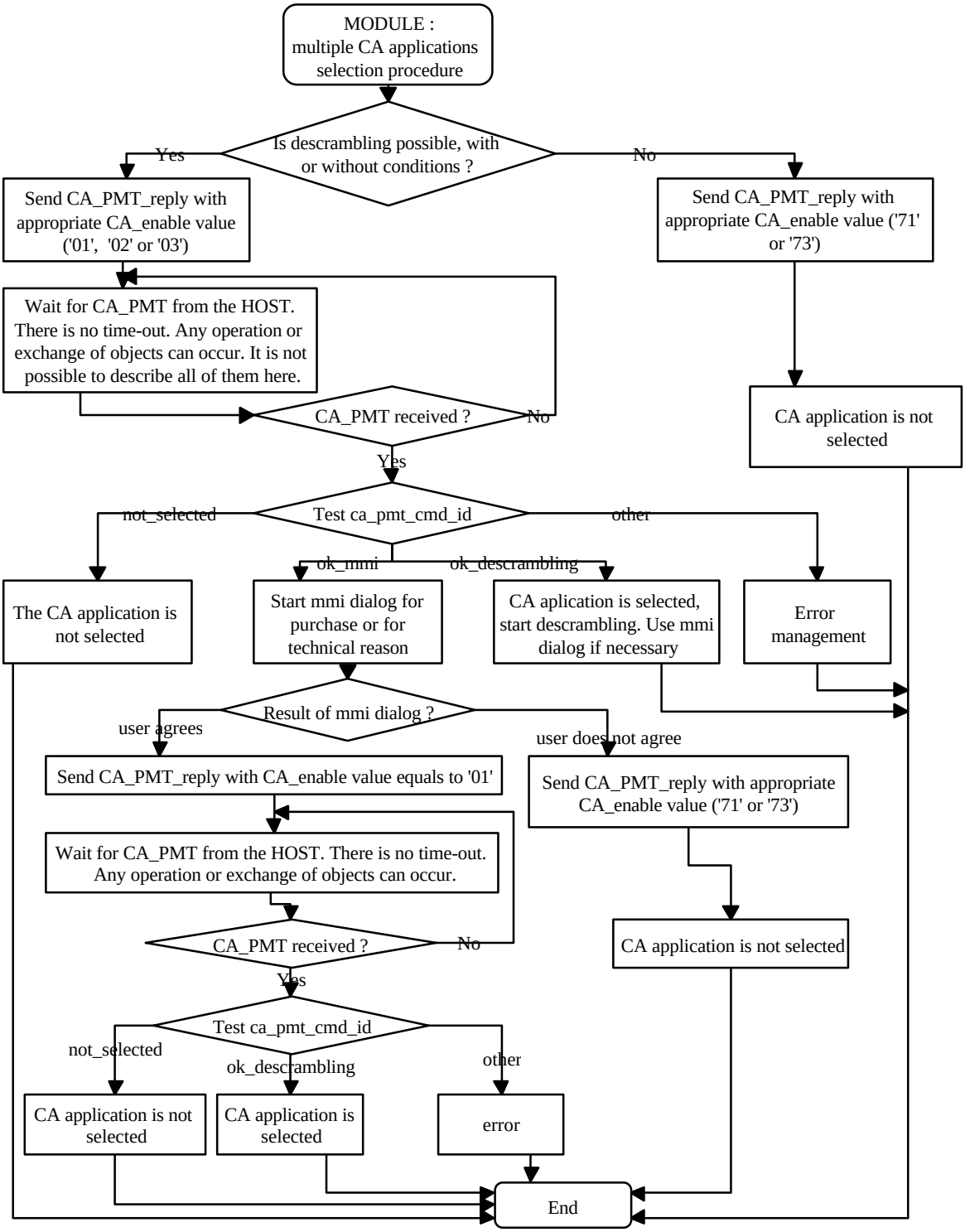


The CA application is considered to be selected, even if there is no entitlement or if there is a technical problem. The mmi dialog is completely handled by the CA application


HOST :
Multiple CA application selection procedure
Send CA_PMT(query) to all still selectable CA applications
Wait for CA_PMT_reply as long as there is no change in the selecte streams. Modules cannot start mmi purchase dialog, any operation c exchange of other objects may exist during this period. It is not possible to describe all of them here. In particular, if the user select another programme, the host starts the whole procedure again, witho waiting for the CA_PMT_reply.
CA_PMT_reply received ?
No
Yes
Test CA_enable ?
01 : descrambling possibl
03 : under conditions (technical dialog)
02 : under conditions (purchase dialog)
F1, F3 : not possible
other value
Send CA_PMT (ok_descrambling) to the CA application that sent the reply and CA_PMT(not_selected) to the others
selectable CA application with status "under conditions (technical dialog)"
selectable CA application with status "under conditions (purchase dialog)"
not selectable CA application
Error management
All CA_PMT_reply received ?
No
Yes
The CA application is selected
Call HOST : Selectable CA application(s) management procedur
End


HOST : Selectable CA application management procedure
How many selectable CA applications ?
n>1
n=1
n=0
Send CA_PMT (ok_mmi) to all selectable CA application
Call single CA application selection procedure
There is no selected CA application
Wait for CA_PMT_reply as long as there is no change in the selected stream (each CA application is expected to open a mmi session. Host may not support more than one mmi session simultaneously). Other operations or exchanges of objects can occur during this period. It is not possible to describe all of them here. In particular, if the user selects another programme, the host does not wait CA_PMT_reply and restart the whole selection procedure again.
CA_PMT_reply received ?
No
Test CA_enable ?
01 : possible
71, 73 : not possible
Other
Select this CA application. Send CA_PMT(ok_descrambling) to this application and CA_PMT(not_selected) to all other selectable CA applications which have not given any response yet.
This CA application is not selected
Error. CA application not selected
All CA_PMT_reply received ?
No
Yes
There is no selected CA application
CA application selected
End


MODULE : Enter selection procedure
open session with CA support resource
Wait for CA_PMT from the HOST. There is no time-out. Any operation or exchange of objects can occur. It is not possible to describe all of them here.
Yes
CA_PMT received for a new programme ?
No
Analyse CA_descriptors. Extract and analyse ECM(s) of selected streams.
Test ca_pmt_cmd_id
'ok descrambling'
'query'
'not selected'
Other
There is no other selectable CA application. Descramble all possible streams, use mmi dialog when needed.
The CA application is selected
Call MODULE : multiple CA applications selection procedure
The CA application is not selected
Error management
End


MODULE : multiple CA applications selection procedure
Is descrambling possible, with or without conditions ?
Yes
No
Send CA_PMT_reply with appropriate CA_enable value ('01', '02' or '03')
Send CA_PMT_reply with appropriate CA_enable value ('71' or '73')
Wait for CA_PMT from the HOST. There is no time-out. Any operation or exchange of objects can occur. It is not possible to describe all of them here.
CA_PMT received ?
No
Yes
CA application is not selected
Test ca_pmt_cmd_id
not_selected
ok_mmi
ok_descrambling
other
The CA application is not selected
Start mmi dialog for purchase or for technical reason
CA aplication is selected, start descrambling. Use mmi dialog if necessary
Error management
Result of mmi dialog ?
user agrees
user does not agree
Send CA_PMT_reply with CA_enable value equals to '01'
Send CA_PMT_reply with appropriate CA_enable value ('71' or '73')
Wait for CA_PMT from the HOST. There is no time-out. Any operation or exchange of objects can occur.
CA_PMT received ?
No
Yes
CA application is not selected
Test ca_pmt_cmd_id
not_selected
ok_descrambling
other
CA application is not selected
CA application is selected
error
End

APPENDIX B: PHYSICAL LAYER DEADLOCK DISCUSSION

This is an example of a particular module single-buffer implementatrion and an argument that it is free of deadlock. No guarantee is given that the particular implementation is robust, or indeed fully functional. It is purely for illustration.

There is a single buffer. This is free if and only if FR==1. The module implements three flag bits: DA, FR and FRx. Only DA and FR are visible to the host. The rules for operation of FR and FRx are:
If FRx is set by the module, both FR and FRx are set together.
If FRx is reset by the module, or by a first byte write from the host, then both FRx and FR are reset together.
If FR is reset by the module only FR is reset, and FRx retains its previous state.
If FR is set by the module then the set operation will not succeed if FRx is reset.

The following discussion considers all possible cases of ordering between host actions and module actions when both are competing to acquire a free buffer, that is, when FR==1. Important timing points in the process are labelled thus: (a1), (a2), (b1), etc.

Host to Module protocol

(a1) Host sets HC -> 1
(a2) Host tests FR
FR==0: (a3) Host sets HC -> 0. end. (Buffer in use)
FR==1: Host can proceed - host writes size, bytes. (a4) FR, FRx -> 0 on first byte write. End of transfer (a5) Host sets HC -> 0. end.

Module to Host protocol

(b1) Module resets FR.
(b2) Module tests FRx.
FRx==0: Module waits. end. (Buffer in use)
FRx==1: (b3) Module tests HC.
HC==1: (b4) Module sets FR and waits. end. (set will fail if FRx is reset - host writing data)
HC==0: Module can proceed - module resets FRx, fills buffer & (b5) sets DA. end.

If the module wants to start a Module to Host cycle there ate the following possibilities:

a) Module resets FR & tests HC before a1. The module sees FRx==1 and HC==0 and will proceed. The host sees FR==0, resets HC and waits a physical layer poll cycle. As the module has proceeded the host must test DA periodically & read the data before FR can become set again.

b) Module resets FR before a2 & tests HC after a1. The module sees FRx==1 and HC==1, attempts to set FR and waits. The host has two possibilities. If the module sets FR again before a2 then the host will proceed. If not, the host sees FR==0, resets HC and waits a physical layer poll cycle. In this case if the module poll cycle is longer than the time taken for the host to reset HC then it will get to start start its cycle next time. Otherwise it may wait one or two module poll cycles. If the module triggers on the host resetting HC it will start its cycle as soon as that happens. The module will proceed first and the host must wait for DA to be set and read the data before FR becomes 1 again. If the module poll cycle is longer than the host's then the host will get to start its cycle first.

c) Module tests FR after a2, tests FRx before a4. The module sees FRx==1. The module *must* then test HC before a5, so HC==1. The module attempts to set FR and waits its own poll cycle. The host will test FR==1 and proceed. The module will need to test for a full buffer and read the data before a new cycle can start. The constraint that the module must test HC before a5 is not onerous since it has already tested FRx before a4, so that it has a complete host buffer write cycle (at least 3 bytes) in which to do it.

d) Module tests FRx after a4. The module sees FRx==0 & waits its poll cycle. A host write cycle is proceeding. The module will need to test for a full buffer and read the data before a new cycle can start.

No other cases occur. Looking from the host's point of view the situation is more or less a mirror image of the above. There are the following possibilities:

e) Host sets HC bit, tests FR bit before b1, cycle proceeds to a4. The host sees FR==1 and proceeds. The module sees FRx==0 and waits its own poll cycle (equivalent to (d) above).

f) Host sets HC bit & tests FR bit before b1. The host sees FR==1 and proceeds. The module sees FRx==1 and HC==1 and waits its own poll cycle (equivalent to (c) above).

g) Host sets HC bit before b3 and tests FR bit after b1. The host sees FR==0, resets HC and waits a physical layer poll cycle. The module sees FRx==1. If it tests HC before the host resets it then it will wait its own poll cycle. if it tests HC after the host resets it then it will proceed (equivalent to (b) above).

h) Host sets HC bit after b3. The host sees FR==0, resets HC and waits a physical layer poll cycle. The module sees HC=0 and proceeds (equivalent to (a) above).

Once either side has gained access to the buffer then FR==0 and remains so until the buffer has been emptied. In this case the following possibilities occur:

a) The host gained buffer access. FR, Frx are reset on the first byte write and the host resets HC when it has finished filling the buffer. In this state DA, Frx and HC are all zero. The module recognises this as the 'host finished writing' condition and can proceed to empty the buffer. When it has done so it sets FR, FRx to 1. If the host sets HCand tests FR during the process, then FR==0 so the host does nothing. If HC happens to be 1 when the module tests it, it will just have to wait another module poll cycle. If the host is polling when the module empties the buffer the module will set FR. The host may or may not see this, but there is no deadlock.

b) The module gained buffer access. The module resets FRx, fills the buffer and sets DA. The host will test DA on a poll cycle and then read the data. DA is reset when the first byte is read, and the host completes the buffer read. The module detects this and sets FR, FRx.

It is important that hosts test the DA bit as well as the FR bit when wishing to transmit, as a single buffer filled by the module will remain busy with DA set and FR reset until the host empties it.