SLOVENSKI STANDARD
SIST R206-001:1999

01-april-1999

Guidelines for the implementation and use of the common interface for DVB decoder applications

Guidelines for implementation and use of the common interface for DVB decoder applications

Ta slovenski standard je istoveten z: R206-001:1998

ICS:

33.160.99	Druga avdio, video in avdiovizuelna oprema	Other audio, video and audiovisual equipment

SIST R206-001:1999 **en**

CENELEC

R206-001

REPORT

July 1998

English version

Guidelines for implementation and use of the common interface for DVB decoder applications

This CENELEC Report has been prepared by the Technical Committee CENELEC TC 206, Consumer equipment for entertainment and information and related sub-systems. It was approved by the Technical Committee on 1996-12-18 and endorsed by the CENELEC Technical Board on 1997-03-11.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Czech Republic, Denmark, Finland, France, Germany, Greece, Iceland, Ireland, Italy, Luxembourg, Netherlands, Norway, Portugal, Spain, Sweden, Switzerland and United Kingdom.

CENELEC

European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

Central Secretariat: rue de Stassart 35, B - 1050 Brussels

Ref. No. R206-001:1998 E

Foreword

This technical report was prepared by the Technical Committee CENELEC TC 206, Consumer equipment for entertainment and information and related sub-systems. TC 206 considered and approved this technical report on 1996-12-18.

It was approved for publication by the CENELEC Technical Board on 1997-03-11.

This technical report is to be read in conjunction with EN 50221:1997, Common interface specification for conditional access and other digital video broadcasting decoder applications.

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, EN 50221:1997.
[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:

ìA 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 formulaî

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

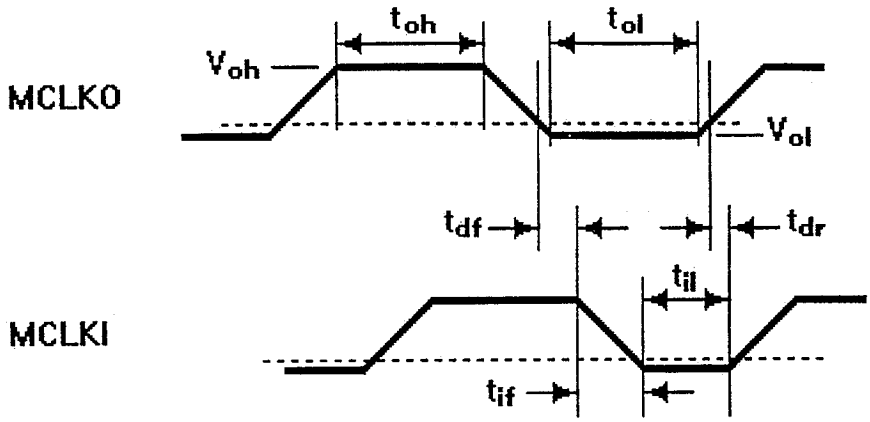
The PC Card electrical specification mainly considers memory or slow I/O cards. So, it does not consider fast clock signals such as those used by the Common Interface. This guideline addresses the specification of these signals.

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 40 ns. 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 40 ns and a period of 111 ns. 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}=2,4 V and V_{ol}=0,5 V 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 0,5 V and V_{oh} is 2,4 V.
- Monotonic between V_{ol} and V_{oh}.
- Minimum low (at V_{ol}) and high (at V_{oh}) interval is 40 ns.
- Maximum transition time between V_{ol} and V_{oh} is 20 ns.
- Minimum transition time between V_{ol} and V_{oh} is 5 ns (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 (111 ns), as the transition time must be less (but not under 5 ns 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 40 ns would make the host buffering requirements too exacting.

Host clock output MCLKI:
- V_{ol} is 0,5 V and V_{oh} is 2,8 V.
- Monotonic between V_{ol} and V_{oh}.
- Minimum low (at V_{ol}) and high (at V_{oh}) interval is 20 ns.
- Maximum transition time between V_{ol} and V_{oh} is 20 ns.
- Minimum transition time between V_{ol} and V_{oh} is 5 ns (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 20 ns (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 50 pF and minimum 10 pF.

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

Under and overshoot

On all signals at all times:
$-0,5\ V <= V_o <= Vcc + 0,5\ V$

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.

Annex 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 300 ms on all messages sent to the module. The module should apply a timeout of 300 ms 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.