Digital Video Broadcasting

Extensions to the Common Interface Specification

DVB Document DVB TM2088r3, CIT 037r11

May 1999

Extensions to the Common Interface Specification

DVB Document DVB TM2088r3, CIT 037r11

May 1999

Version: DVB TM2088r3, CIT 037r11
Last Updated: 20 May 1999 16:22
Date Printed: 20 May 1999 16:22
File: blue_book_ci_front.fr5

1 Introduction and scope

This specification presents a set of extensions to the command interface protocols of the common interface standardised in EN 50221. These provide facilities to allow a diverse range of functions to be delivered to receivers through modules attached to the common interface. In summary the functions supported are:

Input Modules	page 23	Allows modules to deliver transport streams and services to hosts
Status Query Functions	page 45	Allows modules to interrogate the current status/configuration of the host. For example this generic function can be used to implement modules to provide: - Audience metering - Audio description
Power manager	page 54	Allows hosts to determine if modules are busy prior to entering a low power consumption stand-by mode.
Event Management	page 57	Allows modules to register timer events with a host to activate a host from a low power consumption stand by mode.
Application MMI	page 61	Allows a module to interact with the user by loading an application on to the host's application execution environment.
Copy protection	page 66	Allows modules (typically those providing CA functions) to control video copy protection features in a host.
Software download	page 69	Allows a CI module to be used as a source of firmware updates to a host by providing a framework within which manufacturer specific firmware loading protocols can be implemented.
CA pipeline resource	page 79	Allows private communication between applications on a receiver hosted API and CA facilities in a module.

Module identification extension

A key technology enhancement introduced here, and required by several of the above functions, is a method for identifying multiple instances of the same resource. This subdivides the resource_type field in the resource identifier into a smaller resource_type field and a resource_instance field. Accompanying this is a method for hosts to assign locally unique non-volatile IDs to modules (See "Extending use of the resource ID type field" on page 13). The protocols that support this module identification are provided by version 2 of the resource manger.

The following resources depend on this enhancement. Hosts and modules that provide or use these resources shall support version 2 of the resource manger:

- Input Modules
- Status Query Functions
- Event Management
- Copy protection
- CA pipeline resource

Table 1 identifies the element that shall provide the Module ID in each case.:

Resource	Resource Provider	Resource User
Input Modules	✓	
Status Query Functions		✓
Event Management		✓
Copy protection	✓	
CA pipeline resource	✓	

Table 1. Requirement for Module ID

1.1 From version 1

A set of standards has been designed to be used in digital video broadcasting. These standards include source coding, channel coding, service information and decoder interfaces. In addition, a conditional access system is used when there is a need to control access to a broadcast service. It has been decided that the conditional access system need not be standardised, although a common scrambling algorithm is provided. It remains for broadcasters to access decoders with different conditional access systems and to ensure that they have choice of supply of such systems. A solution is to use the common scrambling algorithm and to execute solutions for access based on commercial agreements between operators. This solution can operate with single CA systems embedded in decoders.

A second solution is based on a standardised interface between a module and a host where CA and more generally defined proprietary functions may be implemented in the module. This solution also allows broadcasters to use modules containing solutions from different suppliers in the same broadcast system, thus increasing their choice and anti-piracy options. The scope of this document is to describe this common interface.

The decoder, referred to in this specification as the host, includes those functions that are necessary to receive MPEG-2 video, audio and data in the clear. This specification defines the interface between the host and the scrambling and CA applications, which will operate on an external module.

Two logical interfaces, to be included on the same physical interface, are defined. The first interface is the MPEG-2 Transport Stream. The link and physical layers are defined in this specification and the higher layers are defined in the MPEG-2 specifications. The second interface, the command interface, carries commands between the host and the module. Six layers are defined for this interface. An example of a single module in connection with a host is shown in Figure 1.

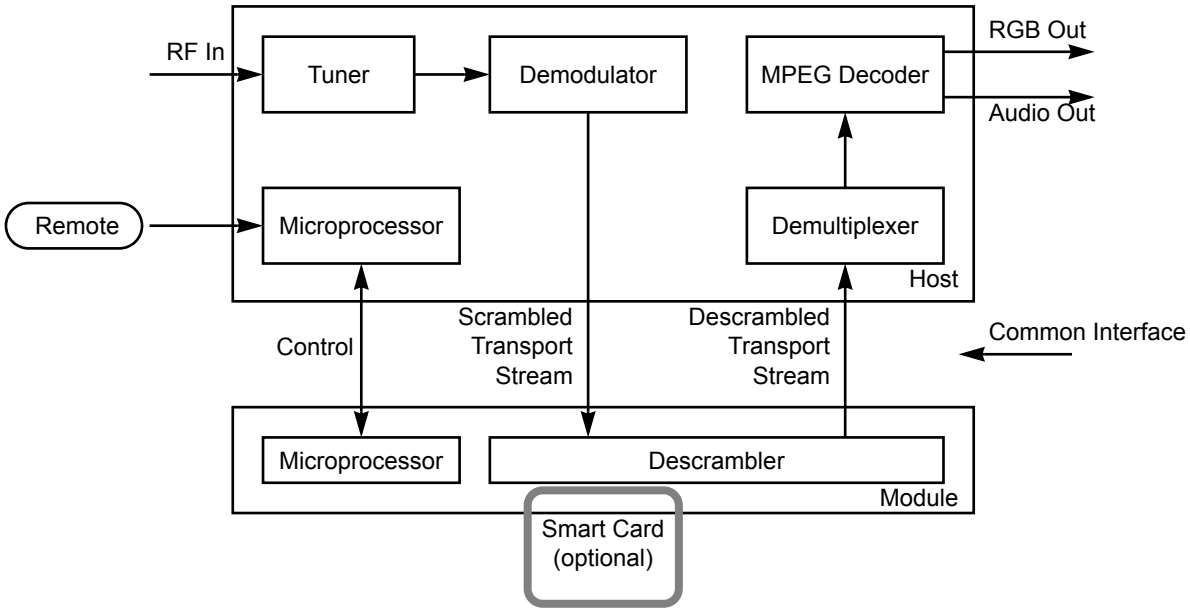

Figure 1. Example of single module in connection with host

This specification only defines those aspects of the host that are required to completely specify the interactions across the interface. The specification assumes nothing about the host design except to define a set of services which are required of the host in order to allow the module to operate.

The specification does not define the operation or functionality of a conditional access system application on the module. The applications which may be performed by a module communicating across the interface are not limited to conditional access or to those described in this specification. More than one module may be supported concurrently.

2 Definitions and Abbreviations

CATV	Community Area TV
Conditional Access **CA**	A system to control subscriber access to services, programmes and events e.g. Videoguard, Eurocrypt.
CENELEC	European Committee for Electrotechnical Standardisation. Central Secretariat: rue de Stassart 35, B - 1050 Brussels.
CI	Common Interface
DVB **Digital Video Broadcasting**	DVB Project Office, c/o European Broadcasting Union, 17 A Ancienne Route, CH-1218 Grand-Saconnex, Geneva, Switzerland. Phone: +41 22 717 27 19. Fax: +41 22 717 27 27. Email: dvb@ebu.ch
DVB-C	DVB Cable
DVB-S	DVB Satellite
DVB-T	DVB Terrestrial
EPG	Electronic Program Guide
elementary stream **ES**	ISO/IEC 13818-1 A generic term for one of the coded video, coded audio or other coded bit streams in PES packets. One elementary stream is carried in a sequence of PES packets with one and only one stream_id.
event	A grouping of elementary broadcast data streams with a defined start and end time belonging to a common service, e.g. first half of a football match, News Flash, first part of an entertainment show
event_id	Defined in ETS 300 468.
Event Information Table **EIT**	Defined in ETS 300 468.
EIT_{pf}	Event Information Table, present/following
EIT_{pfo}	Event Information Table, present/following (other)
LNB	Low Noise Block
Man Machine Interface **MMI**	
MHEG	Multimedia and Hypermedia Experts Group.
MPEG	Motion Picture Experts Group
MPEG-2	Refers to the standard ISO/IEC 13818. Systems coding is defined in part 1. Video coding is defined in part 2. Audio coding is defined in part 3.
network	A collection of MPEG-2 Transport Stream multiplexes transmitted on a single delivery system, e.g. all digital channels on a specific cable system.
network_id	Defined in ETS 300 468.
Network Information Table **NIT**	Defined in ETS 300 468.
PMT	Program Map Table
PSI	Program Specific Information

SDT	Service Description Table, defined in ETS 300 468.
SDT_o	Service Description Table (other)
SI	Service Information
	Digital data describing the delivery system, content and scheduling/timing of broadcast data streams etc. It includes MPEG-2 PSI together with independently defined extensions.
SMATV	Satellite Master Antenna TV
TS	A Transport Stream is a data structure defined in ISO/IEC 13818 1 [1] It is the basis
Transport Stream	
VOD	Video On Demand

3 Normative References

[1]	ETS 300 468	Draft EN 300 468 v1.3.1 (1997-09): “Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems”
[2]	ETR 162	Digital broadcasting systems for television, sound and data services; Allocation of Service Information (SI) codes for Digital Video Broadcasting (DVB) systems.
[3]	ETR 211	ETR 211 (Second Edition, August 1997): “Digital Video Broadcasting (DVB); Guidelines on implementation and usage of Service Information (SI)”
[4]	EN 50221	Common Interface Specification for Conditional Access and other Digital Video Broadcasting Decoder Applications
[5]	ISO/IEC 13818-6	Information technology - Generic coding of moving pictures and associated audio information: Extensions for Digital Storage Media Command and Control.
[6]	R206-001	Guidelines for Implementation and Use of the Common Interface for DVB Decoder Applications
[7]	ISO/IEC 13522-5	MHEG-5 Information technology - Coding of multimedia and hypermedia information: Support for Base-Level Interactive Applications.

4 Command Interface - Resource Management

4.1 Extending use of the resource ID type field

Some of the additional resources identified in "Command Interface - Additional Resources" require a method by which applications can identify a specific instance of a resource amongst several instances of the same resource class. For example, to allow an EPG to discriminate between several identical input modules each connected to a different network. For these new resources this is addressed by allocating the 6 least significant bits of the type field as a resource instance field.

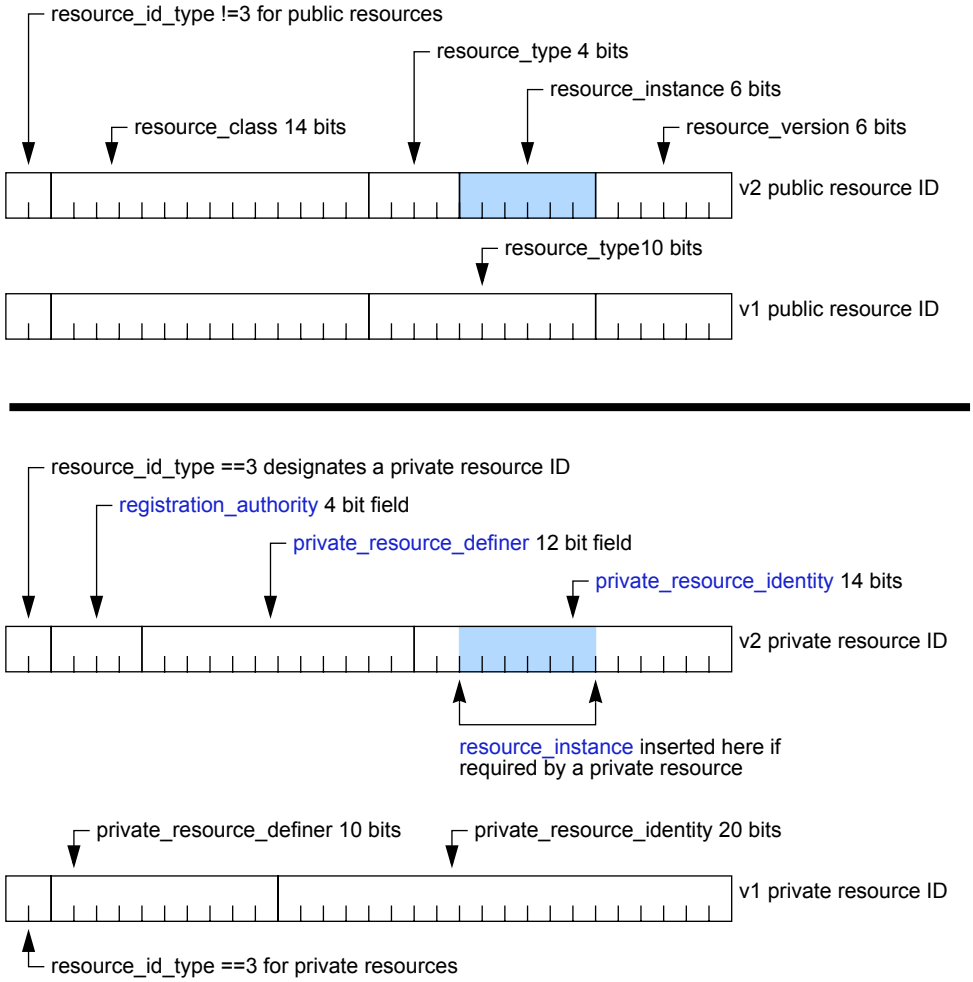

Figure 2. Resource ID coding

Instances of the same resource are discriminated by the lower 6 bits of what was previously the resource_type field. Resources discriminated in this way are advertised by modules and the host during the profile enquiry/reply dialogues during initialisation. Later, a specific instance of a resource can be accessed by opening a session to the resource with that resource ID in the normal way.

In the set of resources developed in this specification 3 uses of resource discrimination are seen:

1. Discriminating identical resources provided by modules

   Here modules use a host allocated Module ID to fill the resource_instance field of the resources that they declare.

   This case applies, for example, to input modules. Instances of identical input modules, providing connection to different networks by their resource_instance field.

2. Discriminating between host and module provided resource instances

   After the modules have declared their resources (where appropriate using the Module ID in their resource_instance field) the host can allocate other resource_instance values to discriminate host provided instances of the same resources.

   This case might apply where both the host and modules provide instances of a copy protection resource. The host creates resource_instance values for its copy protection resources to avoid collision with those provided by modules.

3. Module specific resource interfaces

   Where a host resource must provide a specific channel of communication with each module using its services the host can declare a “personal” instance of the resource for each module by using the module’s Module ID to in the resource_instance field of the resources it declares.

   This case applies, for example, to the event manager. After the host has determined the set of modules supporting a Module ID the event manager declares an instance each dedicated to one module.

4.2 Establishing the Module ID

Version 2 of the resource manager protocol, in a backwardly compatible way, manages the assignment of a unique identity - Module ID - to each transport connection requiring this functionality.

Module IDs are assigned by the host before resource profiling. They are exhibited by certain resources as part of their resource type field when they declare their resource profile to the resource manager. In this way identical modules presenting identical resources will present distinct resource identifiers. Once assigned, it is recommended that the Module ID is retained by the module in a non-volatile way (i.e. the Module ID is preserved even if power is removed from the module). The host can update the Module ID if required.

The text that follows updates that in section 8.4.1 of EN 50221 to describe the behaviour of the version 2 resource manager.

The following resources depend on this enhancement. Hosts and modules that provide or use these resources shall support version 2 of the resource manger:

- Input Modules
- Status Query Functions
- Event Management
- Copy protection
- CA pipeline resource

Table 2 identifies the element that shall provide the Module ID in each case.:

Resource	Resource Provider	Resource User
Input Modules	✓	
Status Query Functions		✓
Event Management		✓
Copy protection	✓	
CA pipeline resource	✓	

Table 2. Requirement for Module ID

4.2.1 Resource Manager - Version 2

The Resource Manager is a resource provided by the host. There is only one type in the class and it can support any number of sessions. It controls the acquisition and provision of resources to all applications. A symmetrical communication protocol is defined between the module and the host to determine the resources each can provide. The protocol is used first by the host to interrogate each transport connection in turn to determine what resources, if any, are presented for use on that transport connection. Then it is used by applications to find out the total resources available. It is then used periodically when resources change to update the common view of available resources.

The Resource Manager is provided by the host and cannot be superseded by a resource on a module. Any attempt to provide a Resource Manager resource by a module shall be ignored by the host.

4.2.1.1 Resource Manager Protocol

This protocol is in two parts - ModuleID establishment and Resource Profile establishment and notification. The second part is identical to version 1 of the Resource Manager protocol. The first part is a new addition.

When a module is plugged-in, or the host is powered up, one or perhaps two transport connections are created to the module serving an application and/or a resource provider. The first thing an application or resource provider does is to request a session to the Resource Manager resource, using either the version 1 or version 2 variant of the resource ID as appropriate. On successful establishment of the session the Resource Manager sends a Profile Enquiry to the application or resource provider.

Newer modules in older hosts

If a newer module attempts to open a session to the resource manager using the version 2 resource manager resource ID (0x00010042) the standard behaviour of a host that only supports version 1 should be to reply with open_session_response having session_status = 0xF2 (“session not opened, resource exists but version lower than requested”). The subsequent behaviour of the module is implementation dependant for example, the module might open a version 1 session to the host and then present a reduced set of resources.

Unless the resource manager has version ≥ 2 the module shall:

- Omit the Module ID establishment part of the protocol entirely
- Not declare resources that depend on the availability of a Module ID

4.2.1.2 Module ID establishment

On receiving Profile Enquiry a module respecting version 2 of the protocol shall reply with Module ID Send:

1. If the module already has a previously allocated ModuleID (stored by the module in non-volatile form), it returns this in the Module ID Send object.

   If ModuleID has not been previously allocated then a ModuleID value of 0 is sent.

2. The Resource Manager replies with a Module ID Command object. If the command field in this object is set to Acknowledgement, then it accepts the ModuleID as allocated and the module then continues with the Resource Profile establishment phase.

   If the command field in the Module ID Command object is set to Set_ModuleID, then the module_id field contains a new ModuleID. The module responds with a further Module ID Send object with the new ID. The host in turn responds with a Module ID Command acknowledgement, and the Profile protocol can continue as before.

If at some later time the host needs to change a ModuleID, it sends Module ID Command to update the ModuleID, expecting a Module ID Send in response, and acknowledging that. For simplicity, the Resource Profile establishment phase after a Profile Change notification from a module shall be preceded by the Module ID establishment phase, but a Profile Enquiry initiated by the module to the Resource Manager never needs it.

Once a Module ID is established for a module by the above procedure, then this ID shall be used in the Resource Type field of all resources which use the Module ID mechanism for distinguishing resource instances.

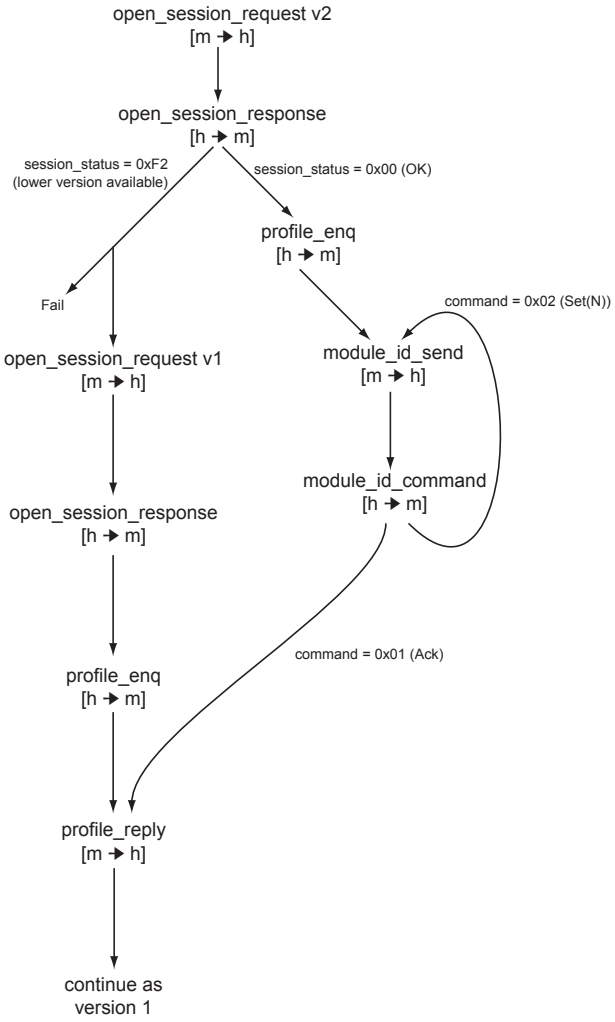

Figure 3. Module ID establishment

4.2.1.3 Resource Profile establishment

Directly following Profile Enquiry in the case of version 1, or following the Module ID establishment phase in the case of version 2, the module sends a Profile Reply listing the resources it provides (if any). The application or resource provider must now wait for a Profile Change object. Whilst waiting for Profile Change it can neither create sessions to other resources nor can it accept sessions from other applications, returning a reply of 'resource non-existent' or 'resource exists but unavailable' as appropriate.

When it has asked for profiles on all transport connections and received Profile Replies the host builds a list of available resources. Where resources have a version 2 resource identifier coding (see Figure 2) multiple instances of the same class & type of resource are automatically differentiated by their resource_instance.

Those resources which have a version 1 resource identifier coding have no discriminating resource_instance. In this case the following rules from EN 50221 apply:

1. Where two or more resources match in both class and type the host keeps the one with the highest version number in its list.
2. Where the version numbers match also the host keeps all resources and chooses one at random when a create session request is received for it.

Once the host has built its resource list it sends a Profile Change object on all current Resource Manager sessions, and those applications that wish to can then ask the host for its list of resources using the Profile Enquiry object.

When it receives the Profile Change notification for the first time the application or resource provider can interrogate the host with a Profile Enquiry and receive a Profile Reply with the host's list of available resources. After this first operation of the Profile Changed protocol the application or resource provider is now free to create or accept other sessions. Its session to the Resource Manager persists to allow further Profile Changed notification by the host from time to time.

If a resource provider wishes to notify a change in the profile of resources it provides, it issues a Profile Changed to the host. The host replies with a Profile Enquiry to which the resource provider replies in turn with its updated resource list. The host processes this and, if this results in any change to the host's own resource list, the host will issue a Profile Changed on all active Resource Manager sessions. The applications can then enquire and receive an updated resource list if they wish.

4.2.1.4 Profile Enquiry

The profile enquiry object requests the recipient to reply with a list of the resources it provides in a Profile Reply object.

Syntax	No. of bits	Mnemonic

`profile_enq () {`
`  profile_enq_tag`
`  length_field()=0`
`}`
	24	uimsbf

Table 3. Profile Enquiry object coding

4.2.1.5 Profile Reply

This is sent in response to a profile enquiry and lists the resources that the sender is able to provide.

Syntax	No. of bits	Mnemonic

`profile_reply ()`
`  profile_reply_tag`
`  length_field() = N*4`
`  for (i=0; i<N; i++) {`
`  resource_identifier()`
`  }`
`}`
	24	uimsbf

Table 4. Profile Reply object coding

- Resource identifiers for the minimum set of resources which shall be provided are listed in section 8.8 of EN 50221.
- Further, optional resources are be listed in annexes to EN 50221.
- A set of additional resources are defined in this specification in "Command Interface - Additional Resources".
- Service providers and manufacturers can define additional "private" resources (see 4.3, "Defining and Using Common Interface Private Resources", on page 19).

4.2.1.6 Profile Changed

The Profile Changed object notifies the recipient that a resource has changed. A module would typically use it to notify the host if the availability status of any of its resources had changed (but not just if a resource was in use). The host would modify its own resource list if necessary, and if there was any change it would in turn send a Profile Changed object on all transport connections.

Syntax	No. of bits	Mnemonic

`profile_changed () {`
`  profile_changed_tag`
`  length_field()=0`
`}`
	24	uimsbf

Table 5. Profile Changed object coding

4.2.1.7 Module ID Send

Send the current ModuleID in response to either a Profile Enquiry, or a Module ID Command updating the ModuleID

Syntax	No. of bits	Mnemonic
module_id_send () {		
module_id_send_tag	24	uimsbf
length_field()=1		
reserved	2	bslbf
module_id	6	uimsbf
}		

Table 6. Module ID Send object coding

module_id
This is the Module ID allocated and managed locally by the host. Only the 6 least significant bits are used. The two most significant bits shall be set to zero when assigning this value and shall be ignored when reading it. A Module ID of zero shall be used by the module if one has not already been allocated by the host in a previous transaction. A value allocated by the host shall be retained by the module through removal of power.

4.2.1.8 Module ID Command

Sent as an acknowledgement of a Module ID Send object, or to set or update an existing ModuleID.

Syntax	No. of bits	Mnemonic
module_id_command () {		
module_id_command_tag	24	uimsbf
length_field()=2		
command	8	uimsbf
reserved	2	bslbf
module_id	6	uimsbf
}		

Table 7. Module ID Send object coding

command

command	command value
Acknowledgement	01
Set_ModuleID	02
reserved	other values

module_id
As defined above.

4.3 Defining and Using Common Interface Private Resources

4.3.1 Introduction

The Common Interface (EN 50221) specification provides a capability for defining and using private resources. This section:

- Modifies the private resource coding defined in EN 50221.
- Defines a registration process for private resource identifiers.
- Describes the technical requirements on private resources.

4.3.2 Defining Private Resources

4.3.2.1 Registering the Resource ID

The Resource Identifier is a 32 bit integer. One quarter of the number space is reserved for use by private resources. Table 8 reproduces the resource identifier coding used by resource manager versions 1 and 2 for public resources and private resources (these are also illustrated graphically in Figure 2 on page 13).

Syntax	No. of bits	Mnemonic
resource_identifier() {		
resource_id_type	2	uimsbf
// resource manager v1 public resource coding		
if ((resource_id_type != 3) && (manager version == 1)) {		
resource_class	14	uimsbf
resource_type	10	uimsbf
resource_version	6	uimsbf
}		
// resource manager v2 public resource coding		
else if ((resource_id_type != 3) && (manager version >= 2)) {		
resource_class	14	uimsbf
resource_type	4	uimsbf
resource_instance	6	uimsbf
resource_version	6	uimsbf
}		
// resource manager v1 private resource coding		
else if ((resource_id_type == 3) && (manager version ==1)) {		
private_resource_definer	10	uimsbf
private_resource_identity	20	uimsbf
}		
else if ((resource_id_type == 3) && (manager version >= 2)) {		
registration_authority	4	uimsbf
private_resource_definer	12	uimsbf
private_resource_identity	14	uimsbf
}		
}		

Table 8. resource identifier coding format

resource_id_type
This 2 bit field distinguishes the public 75% of the number space from the private 25%.

Public resource identifiers

resource_class
This 14 bit integer defines the class of a public resource. The set of these identifiers is recorded in EN 50221 and extended in subsequent public extensions to the common interface specification.

resource_type
This field defines related members of a class of a resources. For example, in EN 50221 for the class “low speed communications” different values in this field differentiate types of return channel interface (serial port, PSTN modem etc.).

In version 1 of the resource manager 10 bits were allocated to the resource type. In version 2 this field is sub-divided to accommodate the resource instance field.

resource_instance
This 6 bit field reflects the module ID of the providing module for certain types of module provided resource.

resource_version
This 6 bit field allows compatibly upgraded versions of public resources to be identified. For example, it identifies the upgraded versions of the resource manager and the application information resource.

Private resource identifiers

registration_authority
This 4 bit field identifies the authority that allocates private_resource_definer values to applicants. This field is managed by ETSI. It allows ETSI to delegate authority for managing parts of the range of private_resource_definer values to other registration authorities.

value	organisation
0	ETSI first allocation block, recorded in ETR 162.
1…15	Allocation blocks for future use by ETSI or delegation to other registration organisations, recorded in ETR 162.

Table 9. Resource identifier registration organisations

private_resource_definer
This 12 bit field identifies an organisation that has obtained registration.

registration authority value	private resource definer value	private resource defining organisation
0	0x000 to 0x0FF	Organisations that have a CA_system_id (registered in ETR 162) are automatically allocated a private definer where the least significant byte of the definer is the most significant byte of CA_system_id.
	0x100 to 0xFFF	Registered by ETSI in ETR 162.
1…15	x	Allocation blocks for future use by ETSI or delegation to other registration organisations, recorded in ETR 162.

Table 10. Resource identifier registration organisations

private_resource_identity
This 14 bit field is available for private allocation by organisations that have been allocated a private resource definer value. Organisations are not required to publish their use of this number space. See “Use of module IDs”.

4.3.2.2 Use of module IDs

As with public resources private resources can use the module ID, allocated to the module by resource manager version 2, to allow different instances of identical modules to be discriminated by an application.

When used, all 6 bits of the module ID shall be used and it shall be inserted in the same position in the private resource identifier as the resource_instance field in a public resource identifier. See Figure 2.

4.3.2.3 Resource object definition

The action of all resources is a protocol based on the exchange of objects. Each object comprises a Tag field, followed by a Length field, followed by zero or more bytes of object content. The objects themselves must be defined, and also the object exchange protocols that implement the resource functionality. The Length field is defined in EN 50221.

For reasons of compatibility with early implementation 3 byte tags shall be used by all resources. The tag values in EN 50221 are globally unique within the specification. This is for historical reasons and new tags defined do not need to be globally unique, only locally unique within one resource.

> *Note: Resource implementors should be aware of this - APDU tags are only unique within a resource. Do not assume that APDU tags will be globally unique.*

Private resource developers are not required to register or publish the tags that their objects use.

4.3.2.4 Resource declaration

The entity offering the resource - host or module - must signal resource availability. This involves notifying the Resource Manager (which runs on the host) of the availability of the resource.

In the case of a module-provided resource, the module must, on receiving a transport connection from the host, create a session to the Resource Manager resource and participate in the resource profile establishment protocol.

In the case of a host-provided resource, the mechanism will depend upon the particular host environment. By whatever means, the Resource Manager must acquire a list of all host-provided resources, including any private ones, during the host initialisation phase. This could be by static definition of resources at host system build time, or by dynamic means during initialisation using an internal protocol or an internal operation of the protocol defined in EN 50221.

Private resources do not have a version field that is known to the Resource Manager so, the version selection protocol used by the Resource Manager for public resources and described in EN 50221 does not apply to private resources. In the case of private Resource Identifier clashes, the Resource Manager will arbitrarily choose one of the conflicting resources to make available in its list. See “Use of module IDs”.

> Note: the “Profile Changed” mechanism (see 4.2.1.3 on page 16 &4.2.1.6 on page 17) can be used by either module or host to declare resources after the initial resource declaration phase is complete.

4.3.2.5 Access to man machine interface

Applications rather than resources use the MMI. If a module provided resource requires access to the MMI (e.g. to allow user configuration) it should respond to an Application Info Enquiry from the host with an Application Info object presenting an appropriate application type.

4.3.3 Using Private Resources

4.3.3.1 From Modules

Applications running on modules will create a session to the Resource Manager and acquire information about all resources available, including private resources. In order to use a resource, the application must “understand” the resource protocol. Therefore the application writer must know both the resource identifier and the protocol specification for the private resource. Beyond that, use of a private resource is identical to use of a public resource - the application creates a session to the resource in the normal manner and then operates the protocol.

4.3.3.2 From Hosts

Hosts may for example provide an API to applications supporting a set of functions such as:

Open Session Request	- Open a session to a resource
Close Session Request	- Close a session to a resource
Send Data	- Send an APDU on a session
Receive Data	- Receive an APDU on a session

All the functions must return state information about the success or otherwise of the operation. These functions are sufficient to provide communication to all resources, however particular host implementations may add extra functionality for reasons of performance or application simplicity. For example, there may be additional functions to give direct access to the Resource Manager's resource database, bypassing the Resource manager protocol defined in EN 50221. There may also be additional functions giving more direct access to host-based resources, or host-based resources may be used entirely independently of the Common Interface infrastructure. This is entirely the decision of the host designer.

5 Command Interface - Application Information

5.1 Application Information - Version 2

5.1.1 New application types

Version 2 of the application information resource (with resource ID 0x00020042) extends the set of application_type values that can be coded in the Application Info object. Table 11 defines the extended list.

Application type	application_type
Conditional_Access	01
Electronic_Programme_Guide	02
Software_upgrade	03
Network_interface	04
Accessibility_aids	05
Unclassified	06
reserved	other values

Table 11. Application type coding

Software_upgrade
Modules that upload software to the host to upgrade the software in the host. No specific upload protocol is implied by this application type.

Unclassified
Modules that don't fall into any other category may be “unclassified”.

A new module application type is not usually allocated unless it is likely that a host will have more than one of a type installed.

Audience metering modules are in this type.

Network_interface
Any type of input module (including both types ‘A’ and ‘B’ described in this specification) can present an application of Network_interface type.

Accessibility_aids
Modules that provide a facility to for those with some form of disability or impairment can use this application type.

Audio description modules are in this type.

5.1.2 Unrecognised application type semantics

A host with a version 2 application information resource will understand the full set of application types listed in Table 11. When presented with an unrecognised application type they shall treat them as Unclassified (type 06).

6 Command Interface - Additional Resources

6.1 Input Modules

Two types of input modules are defined ‘A’ and ‘B’. Type ‘A’ is a simple, potentially low-cost module for delivery of broadcast services via DVB-C, DVB-S or DVB-T networks to hosts. Type ‘B’ (See “Type ‘B’ Input Modules” on page 31) supports these types of service and in addition allows other types of service and network to be delivered.

6.1.1 Requirements for both input module types

6.1.1.1 TS format

Where the input module delivers a Transport Stream (TS) to the host the TS itself and the data streams within it shall conform to the appropriate DVB specifications for a broadcast TS. In particular:

- TS, PSI, Audio and Video data shall conform to ETR 154
- SI shall conform to ETS 300 468 and ETR 211

6.1.1.2 TS control

Input modules shall continue to pass the host supplied TS from its Transport Stream Input to its Transport Stream Output until the host opens a session to the control resource (e.g. StreamInput or ServiceGateway) of the module and sends a command requesting the module to deliver a stream/service (e.g. TuneTSReq or GetServiceReq).

When the host requests the module to stop providing the stream/service or closes the session to the control resource the TS output by the module shall revert to being that supplied by the host.

This requirement does not preclude the module also including CA functions to descramble some or all of the data passing through the module.

6.1.1.3 Input module sessions

Module ID derived resource instances

Each input module shall have a Module ID (See “Extending use of the resource ID type field” on page 13) and use this ID in the resource ID of the resources that it provides.

The resource ID for type ‘A’ input modules is of the form `0000 0000 1000 0000 0001 iiii ii00 0001`. The resource ID of a type ‘B’ input module[1] has the form `0000 0000 1000 0001 0001 iiii ii00 0001`. In each case iiiiii is the Module ID of the input module.

Example

An example is illustrated in Figure 14. Here 3 modules are inserted into a host. Two of these modules are input modules which present resource IDs derived from their Module ID. Applications (either host or module resident) can open sessions to each instance of the input module. One module is a CA module which also has a Module ID, but isn’t an input module, so doesn’t present this type of resource.

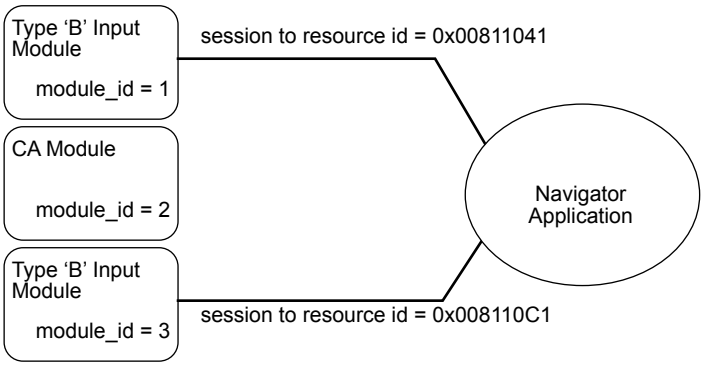

Figure 4. Use of module IDs in input module resource presentation

1. Where its a Broadcast Service Gateway module. Future different types of Service Gateway module will have a different value in the class field.

6.1.2 Type ‘A’ Input Modules

6.1.2.1 Introduction (informative)

Module overview

Figure 5 illustrates a possible type ‘A’ module. Here a low performance microcontroller provides local intelligence within the module. The functions this is will support are:

- User set-up screens, for example, to allow the user to configure a satellite module with regard to the characteristics of the LNB & dish to which it is connected.[1]
- The ability to search for transport streams.
- The ability to tune to transport streams as directed and then remained locked to them.

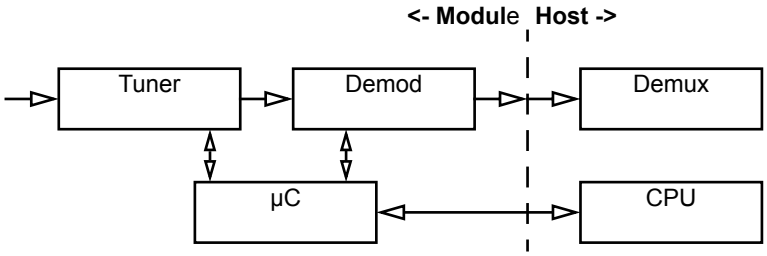

Figure 5. Illustrative type ‘A’ module

Software model overview

Module man machine interface

All input modules shall support host-module communications from the Application Information resource. In particular if a module provides set-up screens these shall be accessible at least in response to Enter Menu message from the host.

Hosts supporting input modules shall provide the user with a method to access the top level menu of each module.

Input Set-up

Depending on the delivery system connected to the module it may be appropriate for the module to provide set-up screens, using the normal CI MMI methods, to assist installers set-up the input to the host. For example, these screens might provide display of signal strength to assist antenna pointing etc.

Autoscan set-up

Depending on the delivery system connected to the module it may be appropriate for the module to provide set-up screens to configure its autoscan process. See below.

Host responsibility

The host has no responsibility in these area other than providing a method for the user to activate the top-level user interface screens of the module.

Scanning for TS

The host is responsible for initiating the frequency scanning process. This applies whether module autoscan feature is used or whether the host directs the scanning in a more hands-on way.

Messages are based on delivery system descriptors

The dialogues between the host and the input module are in terms of the payload of DVB SI delivery system descriptors. For all currently defined DVB delivery systems (DVB-C, DVB-S & DVB-T) this payload is the same size (11 byte) but the internal coding varies.

1. This feature is optional but is likely to be a practical requirement of all real modules.

The host is not required to understand this structure to be able to use the module. However, at the host designer's option, enhanced behaviour may be possible where the host does understand it.

Modules must provide autoscan facilities

All low level input modules shall provide an autoscan function that allows them to search for transmissions.

The module is completely responsible for this process. This specification does not limit how this is done. Various approaches can be illustrated that might fit different circumstances:

- The module is initialised by the supplier with a list of frequencies and the attributes of the dish/LNB with which it works.

  This might be appropriate where a service provider or retailer delivers a "shrink wrap" package of module and dish intended to access a particular service provider.

- The module might be supplied pre-initialised with data on the characteristics of various network operators (e.g. Astra and Eutelsat) and various LNB/dishes. The module must then ask the user to tell it about the circumstances in which it is deployed (e.g. an Acme steerable dish with a Bloggs Inc. "Mark III" LNB)

- The module might provide an "advanced user" set-up. For example, this might provide the user with the ability to configure the method the module should use to select polarization on an LNB (e.g. LNB voltage, 22 kHz tone, DiSEqC etc.)

Module controlled scanning

The host instructs the module to autoscan. Each time the module "finds" a TS it stops scanning and delivers TuningInformationMessage (in data equivalent to a DVB SI delivery system descriptor) to the host. TuneTSReq can be used by the host to request that the TS is delivered to the host. So, the host has an opportunity to store the TuningInformationMessage and to analyse the SI in the TS allowing it to extract service lists etc. Alternatively, the host might just store the TuningInformationMessage and return later to analyse the SI in more detail. The host can tell the module to continue the search. Eventually the module will report "search done".

When the module performs a search for TS the host should not assume that the module has access to the SI within each TS. The host is responsible for analysing the SI in each TS found. For example, in a terrestrial environment the host may be able to get the same TS on more than one frequency. The host is responsible for deciding which set(s[1]) of tuning information to store for each TS.

Host controlled scanning

The host can also construct tuning information to instruct the module to tune. In this way the host can control the search strategy. This gives the host an opportunity to take advantage of any special knowledge it might have. For example, it might "know" about a "barker channel" which provides reliable tuning information. This might allow the host to accelerate tuning.

Features of this type are enabled but not required by this specification. They are therefore an area for product differentiation.

Hosts should be aware that the tuning information provided by the NITs on some delivery systems (e.g. SMATV and Terrestrial) can be unreliable.

Storing tuning information

During the TS scanning process the host will potentially discover many TS and services. It is a host implementation choice to decide how many to remember, and the facilities provided to the user for selecting services.

The minimum information that host needs to retain to be able to return to a TS is the tuning information provided by the module and a reference to the module (to identify it in the case that there is more than one input module in the host). To be able to return to a service the host must at least store the original network ID, the service ID and a reference to the TS holding that service. The quantity of information involved here is likely to be quite modest.

Optionally hosts might store additional information associated with each service such as the service name.

1. It might be useful to store more than one set of tuning information for a TS to accommodate variable reception conditions!

TS & Service selection

Tuning to TS
The host can command the input module to tune to a TS. The TS is specified with the standard 11 byte TuningInformationMessage.

Service Selection
The host is responsible for accessing services within each TS**.** A type 'A' input module is not required to have visibility of the services within a TS.

CA features
Independent of their type 'A' input module functionality, input modules can also provide a Conditional Access Support resource to manage CA access to services within TS. In this case the host communicates independently to the tuning support and the CA support features. Here the host behaviour is almost identical to the case where CA is provided by a separate module down stream of the input module.

6.1.2.2 Type 'A' module command interface

StreamInput

Type 'A' input modules shall present a StreamInput resource to the host. The resource identifier for this resource is 0x00801ii1. This resource shall support a single session.

After a session is opened to its StreamInput resource the module shall continue to pass the host supplied TS from its Transport Stream Input to its Transport Stream Output until TuneTSReq is used to tune to a specified TS when the selected TS replaces the one from the host as the output of the module. The TS output of the module reverts to the TS from the host either when the session to the StreamInput is closed or TuneTSReq is used without a TuningInformationMessage.

Table 12 summarises the set of stream level module control calls presented by the StreamInput resource.

Call	Direction	Description
DeliverySystemInfoReq	h->m	Requests the module to provide information on its delivery system.
DeliverySystemInfoAck	m->h	Reply describing the type of delivery system connected e.g. (DVB-S, -C, -T)
ScanStartReq	h->m	Instructs the module to start scanning for TS
ScanNextReq	h->m	Instructs the module to continue scanning for TS
ScanAck	m->h	Reply describing the TS found
TuneTSReq	h->m	Instructs the module to tune to a TS
TuneTSAck	m->h	Reports the success or otherwise of the tune

Table 12. Overview of the StreamInput objects

DeliverySystemInfoReq
Requests the module to report on the delivery system it connects to.

Syntax	No. of bits	Mnemonic

DeliverySystemInfoReq () {
  DeliverySystemInfoReqTag
  length_field()
}
	24	bslbf

Table 13. DeliverySystemInfoReq syntax

DeliverySystemInfoReqTag
This 24 bit field with value 0x9F8000 identifies this message.

DeliverySystemInfoAck
Reply to DeliverySystemInfoReq describing the type of delivery system connected to the module.

Syntax	No. of bits	Mnemonic
DeliverySystemInfoReq () {		
DeliverySystemInfoAckTag	24	bslbf
length_field()		
for(i=0; i<N; i++) {		
SystemIdentifier	8	bslbf
}		
}		

Table 14. DeliverySystemInfoReq syntax

DeliverySystemInfoAckTag
This 24 bit field with value 0x9F8001 identifies this message.

SystemIdentifier
This 8 bit field identifies the delivery system(s) connected by the module. The values defined for this field defined in Table 15.

SystemIdentifier value	Delivery system	Tuning information message format
0	"Abstract"	Module specific
1	DVB-C	As DVB SI cable delivery system descriptor
2	DVB-S	As DVB SI satellite delivery system descriptor
3	DVB-T	As DVB SI terrestrial delivery system descriptor
> 3	Reserved for future use	

Table 15. Delivery system identification

The TuningInformationMessage format is in all cases 11 bytes long. In cases 1 to 3 the message is the last 11 bytes of the corresponding DVB SI delivery system descriptor (i.e. all bytes after the descriptor tag and length fields). All other delivery systems shall use the same 11 byte format.

Hosts supporting input modules shall be able to work with all delivery systems (even those not yet defined) as there is no requirement for hosts to understand the tuning information message. The purpose in revealing the type of the delivery system is to enable hosts to provide enhanced facilities for delivery systems with which they are "familiar".

"Abstract" delivery systems
The "abstract" delivery system uses a standard size 11 byte tuning information message. However, the coding of this message is not publicly defined.

Example cases where modules may declare their network as "abstract" include:

- SMATV or small CATV networks where the tuning information delivered by the NIT is not reliable following remodulation of signals from a different delivery system.
- New delivery systems with different modulation parameters

ScanStartReq

Instructs the module to start scanning for TS from some "start point" of its own choosing.

On receiving this the module may open a MMI session to request the user to configure parameters affecting the scope of the search. The host shall be able to let the module open a session to the MMI resource.

Syntax	No. of bits	Mnemonic
ScanStartReq () {		
ScanStartReqTag	24	bslbf
length_field()		
}		

Table 16. ScanStartReq syntax

ScanStartReqTag
This 24 bit field with value 0x9F8002 identifies this message.

ScanNextReq

Instructs the module to continue scanning for TS from the "point" achieved when ScanAck last returned.

Syntax	No. of bits	Mnemonic
ScanNextReq () {		
ScanNextReqTag	24	bslbf
length_field()		
}		

Table 17. ScanNextReq syntax

ScanNextReqTag
This 24 bit field with value 0x9F8003 identifies this message.

ScanAck

Reply from the module to the host when a broadcast signal is found, or the search is completed. The TS found is not delivered to the host unless a TuneTSReq is sent.

Syntax	No. of bits	Mnemonic
ScanAck () {		
ScanAckTag	24	bslbf
length_field()		
TSState	8	uimsbf
TuningInformationMessage	11x8	bslbf
ScanProgress	8	uimsbf
}		

Table 18. ScanAck syntax

ScanAckTag
This 24 bit field with value 0x9F8004 identifies this message.

TSState
This 8 bit field delivers an unsigned integer indicating the availability of the TS. The coding of this field is as follows:

- 0 indicates no signal found.

  When auto-scanning for TS '0' indicates that the auto-scan process has searched all possible frequencies.

- 1 to 255 provide a normalised representation of the signal quality (bigger is better).

TuningInformationMessage
This 11 byte field carries a delivery system dependent coding of the tuning information to re-acquire the TS found by the module.

The value of this field is not defined if the TSState is '0'.

ScanProgress
This 8 bit unsigned integer provides an approximate proportional indication of how far through the auto-scanning process the module is. The range of allowed values is 0 to 255. The value increases as the scan progresses.

TuneTSReq

This call requests the module to tune to the TS using the tuning information supplied. If the TuningInformationMessage field is missing (i.e. the length field indicates zero following bytes) then the request is for the module to disconnect from the network.

Syntax	No. of bits	Mnemonic
TuneTSReq () {		
TuneTSReqTag	24	bslbf
length_field()		
TuningInformationMessage	11x8	bslbf
}		

Table 19. TuneTSReq syntax

TuneTSReqTag
This 24 bit field with value 0x9F8005 identifies this message.

TuningInformationMessage
This 11 byte field carries a delivery system dependent coding of the tuning information to acquire the TS found by the module. The coding is identical to the TuningInformationMessage returned by ScanAck.

TuneTSAck

This reply indicates that the module has tuned to the requested frequency in response to a TuneTSReq. The message is sent when the module is delivering a stable TS.

Syntax	No. of bits	Mnemonic
TuneTSAck () {		
TuneTSAckTag	24	bslbf
length_field()		
TSState	8	uimsbf
}		

Table 20. ScanAck syntax

TuneTSAckTag
This 24 bit field with value 0x9F8006 identifies this message.

TSState
This 8 bit field has identical coding to the TSState returned value returned by ScanAck.

In the case that TuneTSReq has no TuningInformationMessage (i.e. the message is "network disconnect") then this field shall have the value '0' (i.e. no signal).

6.1.3 Type ‘B’ Input Modules

6.1.3.1 Introduction (informative)

Module Overview

Figure 6 illustrates a possible type ‘B’ module. This example might be suitable for connections to a broadcast network. In this case the module has a demux and a CPU and hence is able to analyse information about the network (in this case DVB SI) and provide service level access (compared to the TS level access provided by the type ‘A’ module - See “Type ‘A’ Input Modules” on page 25).

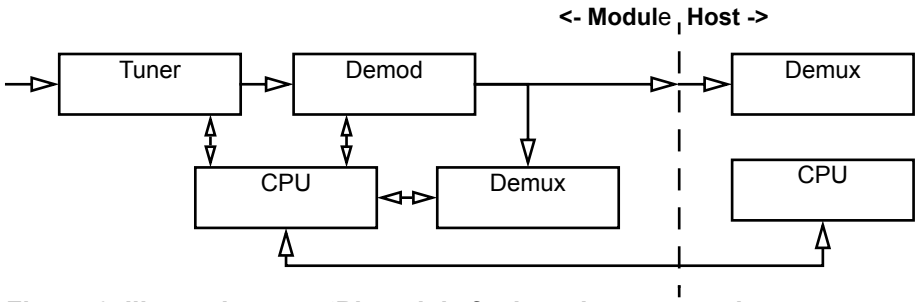

Figure 6. Illustrative type ‘B’ module for broadcast networks

Input modules might also integrate CA functions, in which case Figure 7 might be representative of the module functions required.

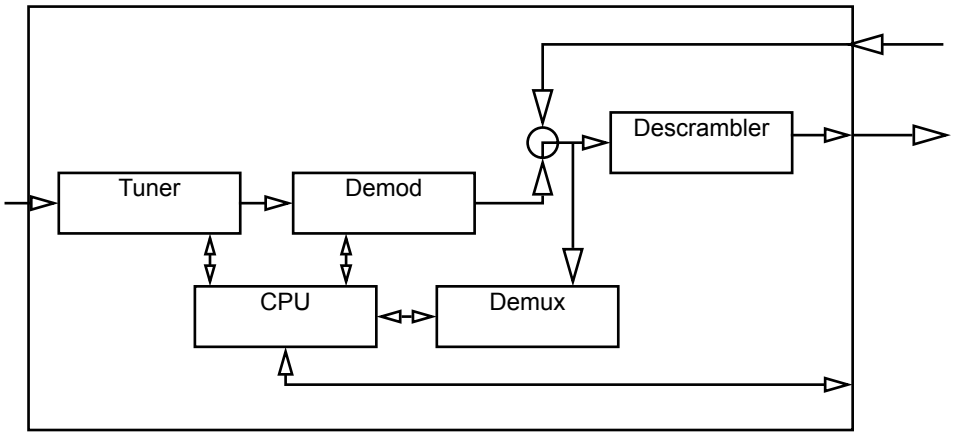

Figure 7. Type ‘B’ input Module with CA

Software Model Overview

This section illustrates possible relationships between the type ‘B’ module and a navigation application.

Navigation model

The model assumes that the basic navigation model for hosts with DVB CI is DVB SI. Hence DVB SI service naming concepts are used by the CI to present the list of available services to the host.

This approach can be used to give generic hosts access to a wide range of TV and other services. However, it is envisaged that in the future this module-host API may evolve and provide new methods that allow “aware” hosts to work more directly with novel service types.

Simple TV access
In Figure 8 a basic host accesses normal TV services provided by a module. In this case the module provides a list of the services it can provide. The application presents this list to the user. When the user selects a service the module delivers the service to the host where it is decoded.

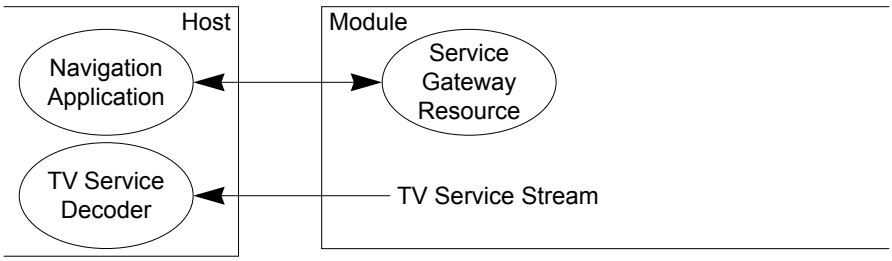

Figure 8. The basic host application / gateway resource relationship

The host's "Navigation Application" can be designed to seamlessly integrate the list of module delivered services with the list of "its own" services.

Basic access to new service types
In Figure 9 the same basic host is connected to a module that can provide access to new types of service. The module provides a decoder for the service and thus insulates the host from having to "understand" the service. For example, the module might be providing access to a wired VOD service. Here, the module resident "Type Specific Decoder" (TSD) is a "browser" which allows the user to navigate the VOD server. The "browser" is presented to the user via the module-to-host Man Machine Interface routines.

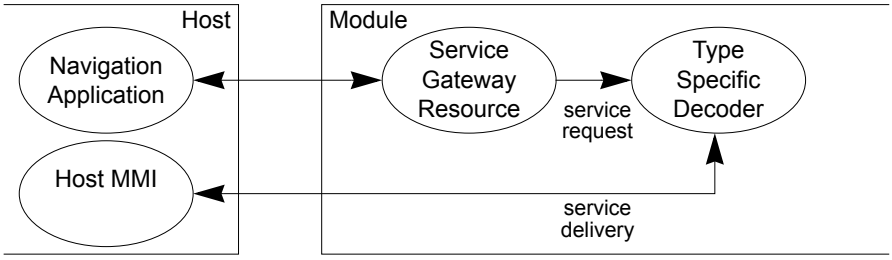

Figure 9. A basic host accessing more advanced services

In Figure 9 the list of services presented by the module includes a service with a new type "Service Gateway". If the user selects this "Service Gateway" service the module activates the TSD. In some cases (e.g. VOD) the TSD allows the user to browse a catalogue. A catalogue selection may result in an MPEG AV stream being sent from the module to the host. In other cases (e.g. home shopping) catalogue browsing may be the end purpose of the TSD.

Aware hosts
In Figure 10 a more advanced host is connected to the module (possibly the same module as in Figure 9). Here the host "recognises" new resource types presented by the module and is able to directly take advantage of them.

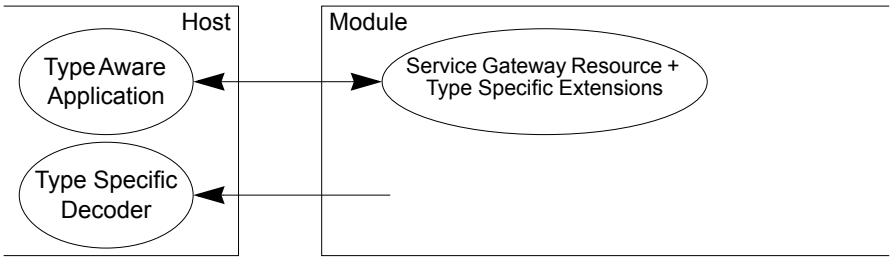

Figure 10. An advanced host application accessing advanced services

An example might be where the module provides access to a network file system (implemented using DSM-CC protocols). The Type Specific Extensions in the module could present the file system by means of the DSM-CC U-U API. The TSD in the host might simple be an MHEG-5/6 engine implemented on top of the DSM-CC U-U API.

Broadcast Type Specific Resource

In this initial proposal the facilities of the basic Service Gateway Resource are outlined (See "Service presentation" on page 34). In addition a set of Type Specific Extensions appropriate to broadcast TV services are outlined (See "Event Presentation" on page 42). These extensions provide information describing the broadcast events which might be of use to host based TV guide.

Evolution of extensions

Typically[1] Service Gateway Modules will present the Generic Service Gateway Resource on a well known resource ID. In addition, the module can also present a Network Specific Service Gateway Resource which inherits the facilities of the generic resource but extends them with facilities specific to the network.

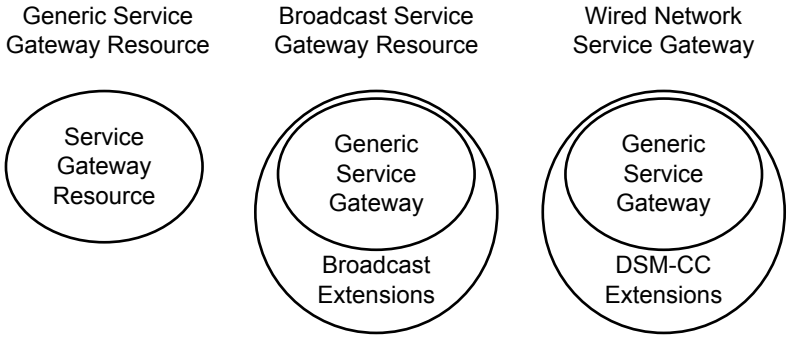

Figure 11. Modules presenting Type Specific APIs

For example, a module providing access to DVB-T broadcasts might also present the variant of the Service Gateway Resource with Broadcast Extensions, allowing it to present event information in addition to service lists.

1. This might not be possible where the types of service accessed are not suitable for presentation via a module resident Type Specific Decoder. In this case the module might only present its more specialist resource.

6.1.3.2 Service presentation

ServiceGateway

Table 21 summarises the set of "Service Gateway" calls presented by the Generic Service Gateway Resource. These facilities are also inherited by ALL other Network Specific Service Gateway Resources.

After a session is opened to its ServiceGateway resource the module shall continue to pass the host supplied TS from its Transport Stream Input to its Transport Stream Output until GetServiceReq is used to request access to a specified service when a new TS may replace the one from the host as the output of the module. The TS output of the module reverts to the TS from the host either when the session to the ServiceGateway is closed or GetServiceReq is used with no service reference.

Call	Direction[a]	Description
ServiceListReq	A->R	Application requests the resource to provide a list of the services that it can supply.
ServiceListAck	R->A	The resource gives the application a list of the IDs of the services that it can provide.
ServiceListVersionReq	A->R	The application request the version number of the resource's service list
ServiceListVersionAck	R->A	The resource provides the version number of its service list
ServiceListChanged	R->A	The resource notifies the application that its service list has changed.
ServiceDescReq	A->R	The application requests further information on a particular service.
ServiceDescAck	R->A	The resource supplies further information on a particular service.
GetServiceReq	A->R	The application requests the resource to provide a service.
GetServiceAck	R->A	The resource replies regarding the availability of a service.

Table 21. Overview of Application<->Resource service interface calls

a] A=host resident application, R=module resident resource. A->R means from application to resource.

ServiceListReq

The ServiceListReq can be issued by the host to request the list of service references that the module can provide. Typically this is done when a module is first configured or each time the host observes that the version number of the service list has changed, but it might also be done in response to a "ServiceListChanged" message from a module.

Syntax	No. of bits	Mnemonic

`ServiceListReq () {`
`  ServiceListReqTag`
`  length_field()`
`}`
	24	bslbf

Table 22. ServiceListReq syntax

ServiceListReqTag

This 24 bit field with value 0x9F8000 identifies this message.

ServiceListAck

In response to "ServiceListReq" a module returns a service list version number followed by a list of the service references that the module can support. These references are persistent as they are either the "real" DVB SI reference to DVB broadcast service or represent a "logical" service that the module can provide.

Changes in the service list presented should represent “significant changes” in the service offering as the host is encouraged to respond by drawing the user’s attention to the change.

Syntax	No. of bits	Mnemonic
ServiceListReq () {		
ServiceListAckTag	24	bslbf
length_field()		
VersionNumber	8	uimsbf
NumberOfServices	16	uimsbf
for(i=0; i<NumberOfServices; i++) {		
OriginalNetworkID	16	bslbf
ServiceID	16	bslbf
}		
}		

Table 23. ServiceListAck syntax

ServiceListAckTag
This 24 bit field with value 0x9F8001 identifies this message.

VersionNumber
This 8 bit integer increments each time the service list is updated.

NumberOfServices
This 16 bit integer giving the number of service references (the value may be 0 if there are no service references).

OriginalNetworkID
This 16 bit field is an original network ID allocated within ETR 162 [2].

ServiceID
This 16 bit field uniquely identifies the service within the original network.

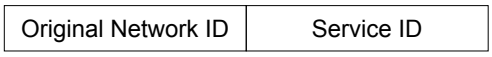

Figure 12. service reference

A transport stream ID is NOT required to uniquely identify a service as is indicated in this extract from section 4.1.1 of ETR 211 [3]:

> A service can be uniquely referenced through the path original_network_id / transport_stream_id / service_id. The network_id, thus, is not part of this path. In addition each service_id shall be unique within each original_network_id.

Host use of service references (informative)
The host use of the “service reference” is not prescribed. However, it is likely that a host that implements concepts such as “favourite channels” will store the “service reference” in non-volatile memory. This allows the host to bind the “service reference” to RCU keys or to organise it into navigator lists according to the user preference.

Hosts might also store other characteristics of the service (e.g. its name) in non-volatile memory. However, this is a optional as the additional data can be requested from the module when required once the host has the “service reference” as an index.

ServiceListVersionReq

The host requests the version number of the service list held by the module. If the version number returned by the module is different from that which the host remembers the host should normally request the service list to investigate the service changes.

This method is provided to give the host an alternative to requesting the full service list each time it is activated.

Syntax	No. of bits	Mnemonic
ServiceListVersionReq () {		
ServiceListVersionReqTag	24	bslbf
length_field()		
}		

Table 24. ServiceListVersionReq syntax

ServiceListVersionReqTag
This 24 bit field with value 0x9F8002 identifies this message.

ServiceListVersionAck

Returns the current version number of the module's list of services in response to ServiceListVersionReq.

Syntax	No. of bits	Mnemonic
ServiceListVersionAck () {		
ServiceListVersionAckTag	24	bslbf
length_field()		
VersionNumber	8	uimsbf
}		

Table 25. ServiceListVersionAck syntax

ServiceListVersionAckTag
This 24 bit field with value 0x9F8003 identifies this message.

VersionNumber
This 8 bit integer increments each time the service list is updated.

ServiceListChanged

Typically the service list presented by a module will be static. A host may scan the list on each activation, this will generally be invisible to the user. The ServiceListChanged message enables the module to inform the host of changes while activated.

For example, this enables module-side service book marks established by the user while interacting with a service gateway to propagate rapidly to the host's navigator service lists (See "Book marks (informative)" on page 40).

Syntax	No. of bits	Mnemonic
ServiceListChanged () {		
ServiceListChangedTag	24	bslbf
length_field()		
VersionNumber	8	uimsbf
}		

Table 26. ServiceListChanged syntax

ServiceListChangedTag
This 24 bit field with value 0x9F8004 identifies this message.

VersionNumber
This 8 bit integer increments each time the service list is updated.

ServiceDescReq

The ServiceDescReq allows the host to request the module to provide more detailed information describing a particular service.

Syntax	No. of bits	Mnemonic
ServiceDescReq () {		
ServiceDescReqTag	24	bslbf
length_field()		
OriginalNetworkID	16	bslbf
ServiceID	16	bslbf
}		

Table 27. ServiceDescReq syntax

ServiceDescReqTag
This 24 bit field with value 0x9F8005 identifies this message.

ServiceDescAck

This message carries the module's reply to ServiceDescReq. The payload is modelled on the parameters of the SDT and the descriptors from the descriptor loop of the SDT of a DVB broadcast service.

Syntax	No. of bits	Mnemonic
ServiceDescAck () {		
ServiceDescAckTag	24	bslbf
length_field()		
OriginalNetworkID	16	bslbf
ServiceID	16	bslbf
reserved_future_use	6	bslbf
EIT_schedule_flag	1	bslbf
EIT_present_following_flag	1	bslbf
running_status	3	bslbf
free_CA_mode	1	bslbf
descriptors_loop_length	12	uimsbf
for(j<0; j<descriptors_loop_length; j++)		
{		
descriptor()		
}		
}		

Table 28. ServiceDescReq syntax

ServiceDescAckTag
This 24 bit field with value 0x9F8006 identifies this message.

reserved_future_use
This 6 bit field has identical meaning to that within the SDT defined in ETS 300 468 [1].

EIT_schedule_flag
This 1 bit field has identical meaning to that within the SDT defined in ETS 300 468 [1].

EIT_present_following_flag
This 1 bit field has identical meaning to that within the SDT defined in ETS 300 468 [1].

running_status
This 6 bit field has identical meaning to that within the SDT defined in ETS 300 468 [1].

free_CA_mode
This 1 bit field has identical meaning to that within the SDT defined in ETS 300 468 [1].

descriptors_loop_length
This 12 bit integer has identical meaning to that within the SDT defined in ETS 300 468 [1].

descriptor()
A descriptor defined for use in the SDT in ETS 300 468 [1] or a private descriptor in the scope of a private_data_specifier_descriptor.

Where the service is a DVB broadcast service the payload shall be the descriptors from its SDT. The minimum requirements for this are described in ETR 211 [3] and ETS 300 468 [1]. Operator's private descriptors may be included using the normal DVB SI methods.

Where the service is **not** a DVB broadcast service at least the mandatory minimum set of SI descriptors shall be used to provide a description of the service. Specifically, these shall convey at least the following:

- CA identifier descriptor (if some part of the service is scrambled)
- Data broadcast descriptor (when required by ETR 211 [3])
- service name (and optionally service provider name) within the service descriptor

Service gateway type
The service type indicated in the service descriptor shall reflect the service type of the broadcast. The service type value 0x0D designates a service of "service gateway type". This will be recorded in ETS 300 468 [1].

Presenting service choices (informative)
As the service description is provided in the same terms as a DVB broadcast service a DVB host should be able to integrate its presentation of the list of service provided by the module with the list of services that the host can provide from its own RF inputs. Hosts can choose whether to cache the module's set of service descriptions into its own RAM (which will make sorting the service lists easier) or whether to request service descriptions from the module as it needs them (which offers a different RAM/processing balance).

Hosts can use the "service gateway" service type information to provide a visual indication that a service gateway is provided. However, it is not essential for hosts to do this.

GetServiceReq

The GetServiceReq requests a module to provide a service. The message payload is service reference. If the service reference is missing (i.e. the length field indicates zero following bytes) then the request is for the module to disconnect from the network.

Syntax	No. of bits	Mnemonic
GetServiceReq () {		
GetServiceReqTag	24	bslbf
length_field()		
OriginalNetworkID	16	bslbf
ServiceID	16	bslbf
}		

Table 29. GetServiceReq syntax

GetServiceReqTag
This 24 bit field with value 0x9F8007 identifies this message.

GetServiceAck

The GetServiceAck message is a response from the module to the host. It informs the host of the progress towards delivering the service and may provide information on which to act.

Syntax	No. of bits	Mnemonic
GetServiceAck () {		
GetServiceAckTag	24	bslbf
length_field()		
OriginalNetworkID	16	bslbf
ServiceID	16	bslbf
Reserved	5	bslbf
ServiceTerminated	1	bslbf
ServiceNotAvailable	1	bslbf
CAServiceFlag	1	bslbf
ActualService	16	bslbf
}		

Table 30. GetServiceAck syntax

GetServiceAckTag

This 24 bit field with value 0x9F8008 identifies this message.

Reserved

These 5 bits are reserved for future use and shall be set to '0'.

ServiceTerminated

This 1 bit field, when set to '1' informs the host that the service has finished (e.g. a VOD film has finished or the user has finished with navigating an information service).

Responsibility for service navigation reverts to the host after this message.

In the case that GetServiceReq has no service reference (i.e. the message is "network disconnect") then GetServiceAck shall return with this field set to '1'.

ServiceNotAvailable

This 1 bit field, when set to '1' informs the host that the service requested is not available. Responsibility for service navigation reverts to the host after this message.

The non-availability of the service may be short-term (i.e. the service is not a full time service and just is not running at the present time) or it may indicate that the service has been deleted. It is the module's responsibility to delete services from its service list if it determines that the service is permanently unavailable.

Depending on the network there may be a concept of "replacement" services. The module is responsible for replacing the requested service with a "replacement" service if this is appropriate for the network.

CAServiceFlag

This 1 bit field, when set to '1' informs the host that conditional access restrictions apply to the service that is being delivered. The host is responsible for using calls such as CA_PMT to obtain access to the service.

This approach will work whether the CA facilities are built into the host, provided in a second CI module (downstream of the input module) or provided by the input module itself. However, in the last case the module may provide a purchasing interface as well as a navigation interface. So, the service may already have been "purchased" by the time it reaches the host. Here the module will NOT indicate that it is delivering a CA service and thus the host won't have to have a CA_PMT dialogue with the module

Allowed flag combinations

Attribute	Allowed combinations		
ServiceTerminated	1	0	0
ServiceNotAvailable	0	1	0
CAServiceFlag	0	0	x
ActualService	0	0	>0

Table 31. Allowed combinations

ActualService
This 16 bit field carries the actual service id of the service being delivered. This allows the module to map “logical” to “actual” services. The value also indicates if the module is delivering a TS/service that the host should decode.

The purpose of the “logical” to “actual” mapping is network dependant, possible uses include:

- Delivering a replacement service when the requested service is not running
- Delivering a replacement service when entitlements for the requested service are not available (CA replacement)
- Translation of a logical “bookmark” service to an actual service (see “Book marks (informative)”)
- Indicating the “actual” service selected from a “navigator” service

Where no TS (or an incorrect TS) is being delivered to the host (e.g. when the service has terminated, the service is not available or the user is still interacting with a navigator provided by the module) the value of actual service shall be zero. This informs the host that it should not attempt to decode a service from the TS.

A non-zero value of actual service indicates that a valid TS is being delivered AND the service ID (MPEG program number) of the service that the host should decode from the TS.

Multiple acknowledges
An input module may generate more than one GetServiceAck in response to a single GetServiceReq. For example, there may be a response with ActualService=0 as the user starts to navigate a service gateway followed by a series of messages with ActualService≠0 as the user selects different service offerings.

The host should assume that the user is interacting with the module until a GetServiceAck message carrying ServiceNotAvailable or ServiceTerminated are set to ‘1’ or the user uses the ‘ESC’ function on the RCU to terminate interaction with the module.

Book marks (informative)
Service lists may exist in several forms, including:

- The list presented by the host navigator to the user
- The list presented by the module to the host
- The list presented by a module’s service gateway to the user while they navigate the service gateway.

Bookmarking (or other methods of identifying a favourite service) are an optional feature of a host service navigator. This document does not seek to comment on how these might be implemented. However, the method by which these might be connected to a module delivered service is described.

The module service gateway may present the user with a service list (as is illustrated in (A) of Figure 13). Alternatively (e.g. in the case of accessing a service server such as VOD) the user may navigate a catalogue of services. In either case the module can provide facilities (such as bookmarks) to allow favourite services to be recalled easily. The set of services presented by the module to the host might be the service gateway and the set of favourite services (as is illustrated in (B) of Figure 13). The host service navigator can then integrate the module service list with its own service list (as is illustrated in (C) of Figure 13).

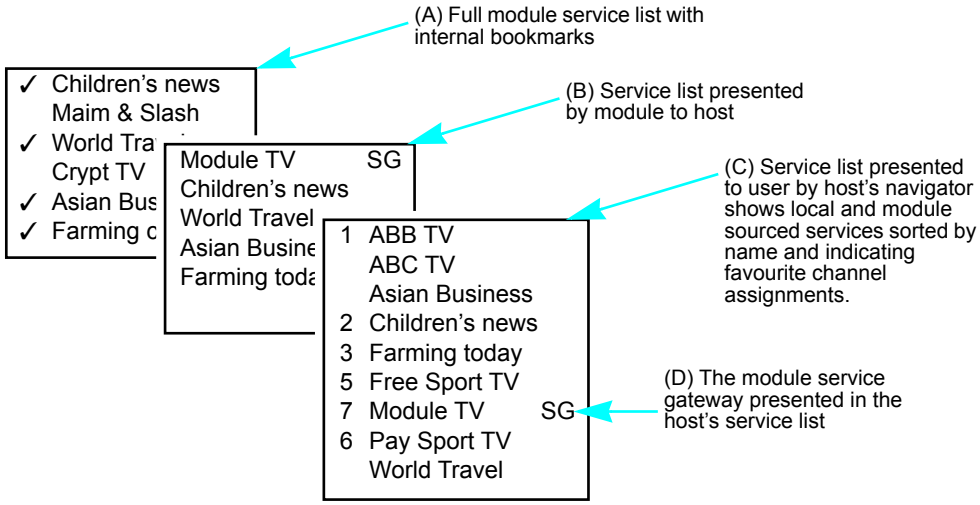

Figure 13. Propagation of module service "book marks" to host lists

The module can issue ServiceListChanged to inform the host of the addition to its service list. The host on discovering this new service should add it to its service lists and could provide the user with a method to "book mark" the service, such as associating it with a key on the RCU.

Peripheral access
(informative)

The service access metaphor has a wide range of application. For example, it could be used to provide access to peripherals connected to the host by a digital network such as 1394. The "service gateway" in this case might provide an interface to control devices such as DVD or DVC. The approach described for Book marks (informative) could be used to allow direct access to DVD discs held in a changer (or "juke box").

6.1.3.3 Event Presentation

In the previous section ("Service presentation" on page 34) the protocols for presenting information on services from a module were described. This approach was generic and so can be applied to many different types of network. This section describes the protocols for presenting information on broadcast events from a module. Events are not a concept that fits all types of network or service. For example, VOD services may have no concept of "next event" or "schedule". Also, even for simple TV services, there may be significant variations in the use of SI between service providers. These will inevitably be reflected in the data that the module can provide to the host.

EITSectionReq

The EITSectionReq requests the module to deliver a specified DVB SI EIT section.

Syntax	No. of bits	Mnemonic
EITSectionReq () {		
EITSectionReqTag	24	bslbf
length_field()		
TableID	16	uimsbf
ServiceID	16	uimsbf
SectionNumber	8	uimsbf
OriginalNetworkID	16	uimsbf
Reserved	7	bslbf
OKToDisruptService	1	bslbf
}		

Table 32. EITSectionReq syntax

EITSectionReqTag
This 24 bit field with value 0x9F8010 identifies this message.

TableID
This 16 bit integer has identical meaning to that within the EIT defined in ETS 300 468 [1]. The set of allowed values, and their definition, are those defined for the EIT (i.e. 0x4E to 0x6F).

ServiceID
This 16 bit integer has identical meaning to that within the EIT defined in ETS 300 468 [1].

SectionNumber
This 8 bit integer has identical meaning to that within the EIT defined in ETS 300 468 [1].

OriginalNetworkID
This 16 bit integer has identical meaning to that within the EIT defined in ETS 300 468 [1].

Reserved
These 7 bits shall be set to '0'.

OKToDisruptService
This 1 bit field, when set to '1', indicates to the module that it is acceptable to disrupt delivery of a current service to obtain the requested event information. If this bit is set to '0' service delivery shall not be disrupted (but the module may not be able to deliver the requested information with this constraint).

EITSectionAck

This returns the parameters of the EIT section requested by EITSectionReq (or an explanation of why it hasn't been provided).

Syntax	No. of bits	Mnemonic
EITSectionAck () {		
EITSectionAckTag	24	bslbf
length_field()		
Reserved	2	bslbf
ResponseCode	2	bslbf
Length	12	uimsbf
for(i=0; i<Length; i++) {		
event_id	16	uimsbf
start_time	40	bslbf
duration	24	uimsbf
running_status	3	uimsbf
free_CA_mode	1	bslbf
descriptors_loop_length	12	uimsbf
for(j=0; j<descriptors_loop_length; j++)		
{		
descriptor()		
}		
}		
}		

Table 33. EITSectionAck syntax

EITSectionAckTag

This 24 bit field with value 0x9F8011 identifies this message.

Reserved

These reserved bits shall be set to '0'.

ResponseCode

This 2 bit field identifies the status of the response:

value	meaning
00	Section not on this TS (but might be available on another TS)
01	Section not available
10	Section found
11	reserved

Table 34. EIT Section response codes

Length

This 12 bit integer specifies the number of bytes following it. This may be zero, see ETR 211 [3].

event_id

This 16 bit integer has identical meaning to that within the EIT defined in ETS 300 468 [1].

start_time

This 40 bit field has identical meaning to that within the EIT defined in ETS 300 468 [1].

duration

This 24 bit field has identical meaning to that within the EIT defined in ETS 300 468 [1].

running_status
This 3 bit integer has identical meaning to that within the EIT defined in ETS 300 468 [1].

free_CA_mode
This 1 bit flag has identical meaning to that within the EIT defined in ETS 300 468 [1].

descriptors_loop_length
This 12 bit integer has identical meaning to that within the EIT defined in ETS 300 468 [1].

descriptor()
Bytes of zero or more descriptors associated with this event.

6.2 Status Query Functions

Status Query

The *Status Query Resource* allows modules to interrogate the status of the host.

6.2.1 Status Query sessions

Module ID derived resource instances

The Status Query resource defines the set of resource instances it shall present after the host has determined the set of modules (and Module IDs) present. One resource instance is created for each Module ID and with the Module ID in its resource_instance field.

Authorised sessions

Some of the information that can be collected by the Status Query resource is potentially private to the user (See "Audience metering" on page 48). The host shall determine that the user is aware of and authorises the data collection. By using a module specific session for each module (which can be authenticated against the modules' ID when the session is opened) the Status Query resource can reserve sensitive information for authorised modules.

Module Connection

Modules requiring the services of the Status Query resource shall open a session to "its" instance of the Status Query resource. I.e. the module derives the resource ID for the instance of the Status Query resource dedicated to serving it using the resource_class defined for the Status Query resource; resource_type =1; resource_instance = Module_ID and resource_version = 1.

Through this module specific session the Status Query resource can send module specific messages (such as StatusAck) to the module.

Example

An example is illustrated in Figure 14 (also compare with Figure 16 on page 57). Here 3 modules are inserted into a host. Two of these modules are CA modules, one implements a module ID and the version 2 resource manager protocols (See "Extending use of the resource ID type field" on page 13). The third module is an audience metering module (See "Selection information" on page 49) and requires the Status Query resource. In this system 2 module IDs are consumed. The host creates 2 instances of the Status Query resource populating the instance field of the resource ID with the module IDs. The module that uses the services of the Status Query resource can open sessions to the appropriate instance of the Status Query resource. The first instance of the Status Query resource remains un-used.

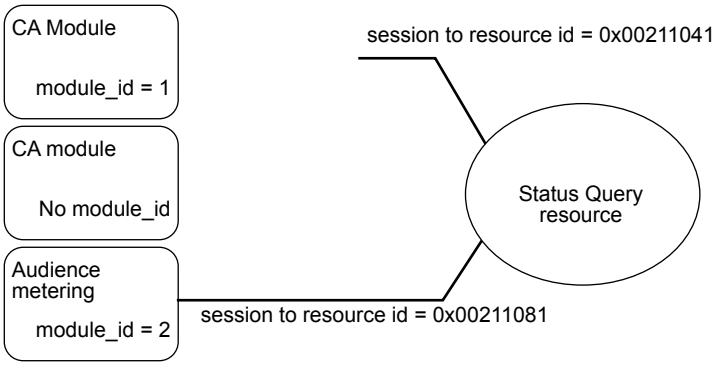

Figure 14. Use of module IDs in Status Query resource presentation

6.2.2 Generic Status Queries

Described here is a facility provided by hosts to provide status information to modules. A generic approach has been taken as it is expected that the set of status items available for interrogation will grow with time.

The host provides a StatusQuery resource which is able to support a session on each of the host-module transport connections. The set of messages for this resource is listed in Table 35.

Message	Direction[a]	Description
StatusQuery(N)	M->H	Requests the host to return the status of status item N.
Trap(N)	M->H	Requests the host to return the status of status item N whenever its value changes.
GetNextItemReq	M->H	The dialogue supported by these calls can be used by the module to explore the set of status items that the host supports.
GetNextItemAck	H->M	
StatusAck	H->M	Returns the status of requested status item as a variable length array of bytes. The format of these bytes will depend on the status item.

Table 35. Messages of the status query resource

a] M=module resident process, H=host's status query resource. M->H means from module to host.

The set of status items that can be interrogated are listed in Table 36.

Status Item Number	Name	Description
0	Reserved	
1	Selection Information	Used to provide Audience Metering Information by describing the inputs and outputs of the host. See "Selection information" on page 49.
2	Port Profile	Also used in Audience Metering, provides a description of the various host ports. See "Port profile" on page 52.
3	Viewed Service	Used to allow an auxiliary decoder (e.g. Audio Description) to track the service being viewed on the host. See "Port profile" on page 52.
4	Activation Status	Describes the power status of the host to the module. See "Activation status" on page 53.

Table 36. List of status items that can be interrogated

6.2.2.1 StatusQuery

Requests the host to report on an item of its status.

Syntax	No. of bits	Mnemonic
`StatusQueryReq() {`		
`  StatusQueryReqTag`	24	bslbf
`  length_field()`		
`  StatusItem`	32	uimsbf
`}`		

Table 37. StatusQueryReq syntax

StatusQueryReqTag
This 24 bit field with value 0x9F8000 identifies this message.

StatusItem
This 32 bit unsigned integer identifies the status item queried. The allowed values, and their definitions are listed in Table 36.

6.2.2.2 Trap

Requests the host to report on changes to the value of a particular status item until the module's session to the StatusQuery resource is closed.

Syntax	No. of bits	Mnemonic
TrapReq() {		
TrapReqTag	24	bslbf
length_field()		
StatusItem	32	uimsbf
}		

Table 38. TrapReq syntax

TrapReqTag
This 24 bit field with value 0x9F8001 identifies this message.

StatusItem
This 32 bit unsigned integer identifies the status item to be monitored. The allowed values, and their definitions are listed in Table 36.

6.2.2.3 GetNextItemReq

Requests the host to return the StatusItem number of the next status item supported by the host after the specified StartStatusItem.

Syntax	No. of bits	Mnemonic
GetNextItemReq() {		
GetNextItemReqTag	24	bslbf
length_field()		
StartStatusItem	32	uimsbf
}		

Table 39. GetNextItemReq syntax

GetNextItemReqTag
This 24 bit field with value 0x9F8002 identifies this message.

StartStatusItem
This 32 bit unsigned integer identifies a start point for a search through the set of supported status items. This value is not required to be one of the status items supported by the host. Typically a module will use the value zero will be used when starting a search.

6.2.2.4 GetNextItemAck

Reply from the host to the module in response to a GetNextItemReq.

Syntax	No. of bits	Mnemonic
GetNextItemAck() {		
GetNextItemAckTag	24	bslbf
length_field()		
NextStatusItem	32	uimsbf
}		

Table 40. GetNextItemAck syntax

GetNextItemAckTag
This 24 bit field with value 0x9F8003 identifies this message.

NextStatusItem
This 32 bit unsigned integer identifies status item number of the first supported status item greater than the StartStatusItem specified in the request.

The value 0 is returned if StartStatusItem is greater than or equal to the status item number of the highest numbered item supported by the host.

6.2.2.5 StatusAck

Reply (from host to module) resulting from a StatusQuery or Trap request from a module. There shall be exactly one StatusAck in response to each StatusQuery. Following a Trap request there shall be a StatusAck in response delivering the current value of the status item, there shall also be a further StatusAck each time the value of the status item changes until the module's session to the StatusQuery resource is closed.

Syntax	No. of bits	Mnemonic
StatusAck () {		
StatusAckTag	24	bslbf
length_field()		
StatusItem	32	uimsbf
for(i=0; i<N; i++) {		
StatusBytes	8	bslbf
}		
}		

Table 41. DeliverySystemInfoReq syntax

StatusAckTag
This 24 bit field with value 0x9F8004 identifies this message.

StatusItem
This 32 bit unsigned integer is the StatusItem value from the a StatusQuery or Trap request that lead to this reply.

StatusBytes
This set of bytes conveys the status information corresponding to the StatusItem. The coding of this information will depend on the status item interrogated.

If the host does not support the status item requested in the StatusQuery or Trap request there shall be an immediate reply with no status byte information.

Table 42 identifies the set of formats for status information defined at the time of writing.

Status Item Number	Definition of status bytes
0	None allowed
1	See Table 43, "Selection information status data," on page 49
2	See Table 48, "Port profile status data," on page 52
3	See Table 49, "Viewed service status data," on page 53
4	See Table 50, "Activation status data," on page 53

Table 42. List of status items that can be interrogated

6.2.3 Audience metering

To support *Audience Metering* for the purpose of market analysis hosts support the following status items:

- Selection information
- Port profile

6.2.3.1 Protecting consumer privacy

Some of the data provided by this status enquiry is private to the consumer. As in normal operation it will not be apparent to the consumer that this data is being collected the host shall ensure that the consumer is aware of, authorises and can discontinue the data collection. The exact method employed is outside the scope of this specification. The method described here is presented as an informative example.

Authorising a module (informative)

The host maintains in non-volatile memory a list of authorised modules. The module identification can be the non-volatile *Module ID* allocated to the module by the host.

When a module not previously authorised first attempts to perform StatusQuery or Trap on the Selection Information status item the host initiates a dialogue with the consumer to establish their willingness for data to be collected before allowing responses to the StatusQuery or Trap to be sent to the module. Once a particular module is authorised the host does not interrogate the consumer again. If the consumer does not authorise the module the host shall close the session thus indicating to the module its rejection.

Consumer control (informative)

A user interface method should be provided by the host which identifies the modules that are authorised and allows the user to deauthorise them.

6.2.3.2 Selection information

The *Selection Information* status data is a list of descriptions of signal sources with their associated destinations. Each input may go to zero or more destinations. The set of input ports shall be complete and shall list each input port only once. If an input port is not connected to an output zero destinations shall be specified.

If Trap() is used to interrogate the selection information a reply will be issued each time the user alters the configuration of the host. The host should only report configuration changes that last for at least 1 second.

Syntax	No. of bits	Mnemonic
time	40	bslbf
while(there is data in the object) {		
in_port_id	8	bslbf
length_in_signal_desc	8	uimsbf
for(i=0; i<length_in_signal_desc; i++) {		
in_signal_desc	8	bslbf
}		
reserved	4	bslbf
length_outputs	12	uimsbf
for(i=0; i<length_outputs; i++) {		
out_port_id	8	bslbf
length_out_signal_desc	8	uimsbf
for(i=0; i<length_out_signal_desc; i++) {		
out_signal_desc	8	bslbf
}		
}		
}		

Table 43. Selection information status data

time

time encoded as in the UTC_time field of the DVB SI Time and Data Table.

in_port_id
identifier of the source of a signal.

in_port_id	description
0 - 7	RF Modulated digital source 0 to 7
8 - 15	IEEE 1394 port 0 to 7
16 - 23	SCART port 0 to 7
24 - 31	CI input module sources 0 to 7
32 - 126	Reserved for future use
127	No source
128 - 255	Manufacturer specific ports

Table 44. In port values

length_in_signal_desc
the number of bytes in the description of the input signal.

in_signal_desc
a block of bytes describing the input signal. The format of this block depends on the input port.

in_port_id	signal source description		
		no bits	mnemonic
0 - 7	DVB SI style delivery system description: original_network_id network_id transport_stream_id service_id Video component tag (0xFF if not found) Audio component tag (0xFF if not found)	 16 16 16 16 8 8	 uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf
8 - 15	<TBD>		
16 - 23	Empty		
24 - 31	CI input module sources 0 to 7.		
	When type 'A': TuningInformationMessage service_id Video component tag (0xFF if not found) Audio component tag (0xFF if not found)	 11x8 16 8 8	 bslbf uimsbf uimsbf uimsbf
	When type 'B': original_network_id service_id Video component tag (0xFF if not found) Audio component tag (0xFF if not found)	 16 16 8 8	 uimsbf uimsbf uimsbf uimsbf
32 - 126	Reserved for future use		
127			
128 - 255	Manufacturer specific string of bytes		

Table 45. In signal description blocks

reserved
this 4 bit field should be set to '0'.

length_outputs
the number of bytes in the description of the output signal(s).

out_port_id
identifier of the destination of the signal.

out_port_id	description
0 - 7	Display 0 to 7
8 - 15	IEEE 1394 port 0 to 7
16 - 23	SCART port 0 to 7
24 - 31	RF Modulator 0 to 7
32 - 126	Reserved for future use
127	No output
128 - 255	Manufacturer specific ports

Table 46. Out port values

length_out_signal_desc
the number of bytes in the description of the input signal.

out_signal_desc
a block of bytes describing the output signal. The format of this block depends on the output port.

out_port_id	signal destination description		
		no bits	mnemonic
0 - 7	Visibility measure 0 -> obscured 1 -> partially obscured 2 -> fully visible >2 reserved	8	bslbf
8 - 15	<TBD>		
16 - 23	Empty		
24 - 127	Reserved for future use		
128 - 255	Manufacturer specific string of bytes		

Table 47. Out signal description blocks

6.2.3.3 Port profile

The *Port Profile* status data provides a textual definition of the host and each input and output port.

Syntax	No. of bits	Mnemonic
receiver_identification_length	8	uimsbf
for(i=0; i< receiver_identification_length; i++){		
receiver_identification_char	8	uimsbf
}		
for(j=0; j<N; j++){		
in_port_id	8	bslbf
length_in_port_desc	8	uimsbf
for(i=0; i< length_in_port_desc; i++) {		
in_port_desc	8	bslbf
}		
out_port_id	8	bslbf
length_out_port_desc	8	uimsbf
for(i=0; i< length_out_port_desc; i++) {		
out_signal_desc	8	bslbf
}		
}		

Table 48. Port profile status data

receiver_identification_length
length of the following string in bytes.

receiver_identification_char
a string of characters (coded according to annex A of DVB SI) uniquely describing the receiver manufacturer, model and version.

in_port_id
see above.

length_in_port_desc
length of the following string in bytes.

in_port_desc
a string of characters (coded according to annex A of DVB SI) describing the input port.

Note: if an input port type defines more than one form of coding of the signal description in table 45 then the port description shall identify the particular form of the encoding used. So, for example, where the signal source is a CI input module the in port description shall at least distinguish if the port is being used as a type ‘A’ input or a type ‘B’ input.

out_port_id
see above.

length_out_port_desc
length of the following string in bytes.

out_signal_desc
a string of characters (coded according to annex A of DVB SI) describing the output port.

Note: if an output port type defines more than one form of coding of the signal description in table 47 then the port description shall identify the particular form of the encoding used.

6.2.3.4 Auxiliary decoder

The *Viewed Service* status item allows a module to implement a decoder in addition to those provided by the host. Typically this might allow modules to implement additional audio decoders to allow the decoding of an audio description stream concurrently with the main programme audio.

Ensuring consumer permission

See "Protecting consumer privacy" on page 49.

Viewed Service

The *Viewed Service* status data indicates the program or components selected by the consumer to be most significant on the display of the host[1].

If Trap() is used to interrogate the viewed service a reply will be issued each time the user changes channel.

Syntax	No. of bits	Mnemonic
service_id	16	bslbf
number_components	8	uimsbf
for(i=0; i<number_components; i++) {		
component_tag	8	uimsbf
}		
}		

Table 49. Viewed service status data

service_id
corresponds to the service_id/program_number of the program currently selected for display by the host. The program number 0x0000 should be used to indicate that the source of the signal applied to the display is not in the Transport Stream available to the module (e.g. the signal source is an analogue VCR connected to a SCART interface).

number_components
number of components tags that follow.

component_tag
the component tag of the component currently selected for decoding by the consumer if a component tag is provided for this component in the PMT by a stream identifier descriptor.

Informative note

Auxiliary decoder modules are expected to be able to parse the PMT of the currently selected service and possibly also certain SI tables. For example, if a service provides soundtracks in more than one language the component tag will identify the audio component currently selected by the consumer. By examining the PMT a module should be able to identify the language of the selected audio stream. An audio description module could then analyse the ISO 639 Language descriptors in the PMT to determine the stream (if any) providing audio description in that language.

6.2.4 Activation status

The *Activation* status data describes the power status of the host.

Syntax	No. of bits	Mnemonic
reserved	4	bslbf
event_activated	1	bslbf
activation_state	3	bslbf

Table 50. Activation status data

1. I.e. the host implements picture-in-picture the dominant window should be considered. If an information service conceals most of the display area the service ID of the information service should be indicated.

reserved
These 4 bits are reserved for future use and shall be set to ‘0’.

event_activated
This 1 bit field, when set to ‘1’ indicates that the host was activated by an event from the event manager (See “Event Management” on page 57) rather than by user action (which is indicated when this bit is set to ‘0’).

When the host has been event activated it is likely that a user is available to respond to dialogues generated by the module.

activation_state
this value identifies the power-up state of the host.

activation state	current power mode
0	Reserved
1	Standby-active[a]
2	On[b]
2 - 7	Reserved for future use

Table 51. Activation state status values

a] Corresponds to the EACEM defined power mode “Standby-active”

b] Corresponds to the EACEM defined power modes “On (play)” and “On (record)”

6.3 Power manager

The Power manager resource, with ID 0x00220041, is a module provided resource that allows a module to indicate to the host that it is engaged in a task that should be allowed to complete.

When one or more modules present the Power manager resource, the host may interrogate each instance of this resource before deactivating the power supply to the modules. If any module is busy the deactivation shall be postponed.

Modules shall continue to operate

Modules shall continue to operate after they have indicated that it is OK for the host to shutdown. For example, a CA module shall continue to descramble data, an input module shall continue to deliver data etc. This operation continues until explicitly stopped by the host (e.g. by the host closing sessions).

Modules can reassert “busy”

If, after a module has indicated that it is OK for the host to shutdown, there is session traffic between the module and the host (either module or host initiated) the host shall ignore any previous indication from the module that it is ready to shutdown. The host should therefore re-interrogate the module before shutting down.

6.3.1 Activation state change request

The Activation state change request object from the host to the module “asks” the module if it is “occupied” with a task that should be allowed to complete before powering-down the host.

Syntax	No. of bits	Mnemonic
`activation_state_change_request() {`		
`  activation_status_change_request_tag`	24	uimsbf
`  length_field()`		
`  reserved`	4	bslbf
`  activation_state`	4	bslbf
`  }`		

Table 52. Activation status state change request object

activation_status_change_request_tag
This 24 bit field with value 0x9F8000 identifies this message.

reserved
These 4 bits are reserved for future use and shall be set to '0'.

activation_state
this value identifies the requested new activation state.

activation state	requested power mode
0	Standby-passive[a]
1 - 15	Reserved for future use

Table 53. Activation state request values

a] Corresponds to the EACEM defined power mode "Standby-passive"

Minimum repetition interval
Hosts should not send Activation state change requests to a module more often than once each minute.

6.3.2 Activation state change acknowledge

The Activation state change acknowledge object is sent in response to a Activation state change request object. It provides an opportunity for the module to indicate that it is performing a task. If any module provides this indication the host shall defer the process of changing the activation state (i.e. it should defer the shutdown). However, modules should not delay the removal of power without good reason.

If a module does not reply within 1 second of an Activation state change request the host can assume that the module assents to the state change.

Syntax	No. of bits	Mnemonic
`activation_state_change_ack() {`		
`  activation_status_change_ack_tag`	24	uimsbf
`  length_field()`		
`  reply_code`	8	bslbf
`  }`		

Table 54. Activation status change reply object

activation_status_change_ack_tag
This 24 bit field with value 0x9F8001 identifies this message.

reply_code
this value identifies the modules response to the requested state change.

reply_code	description
0	OK to change state
1	Module busy, don't change state
2 - 255	Reserved for future use

Table 55. Activation status change acknowledge values

6.3.2.1 Overview of dialogues (informative)

Figure 15 illustrates a possible host/module dialogue sequence to show the use of some of the power management resource calls.

First some event, possibly a timer event, activates the host. This is followed by the host's normal initialisation of its CI. All installed modules can then start work. In this example we focus on module 'A', but module 'B' could also perform some tasks.

As the host was "woken" by a timer event it is "trying to get back to sleep" so, periodically (See "Minimum repetition interval" on page 55) the host polls all modules to see if it can shut down. While module 'A' performs its task(s) it replies "Module busy" when asked, module 'B' replies "OK to change state". As one or more of the modules is busy the host defers going to sleep.

After a period module 'A' completes its work and, like 'B', replies "OK to change state". At this point the host can shut down.

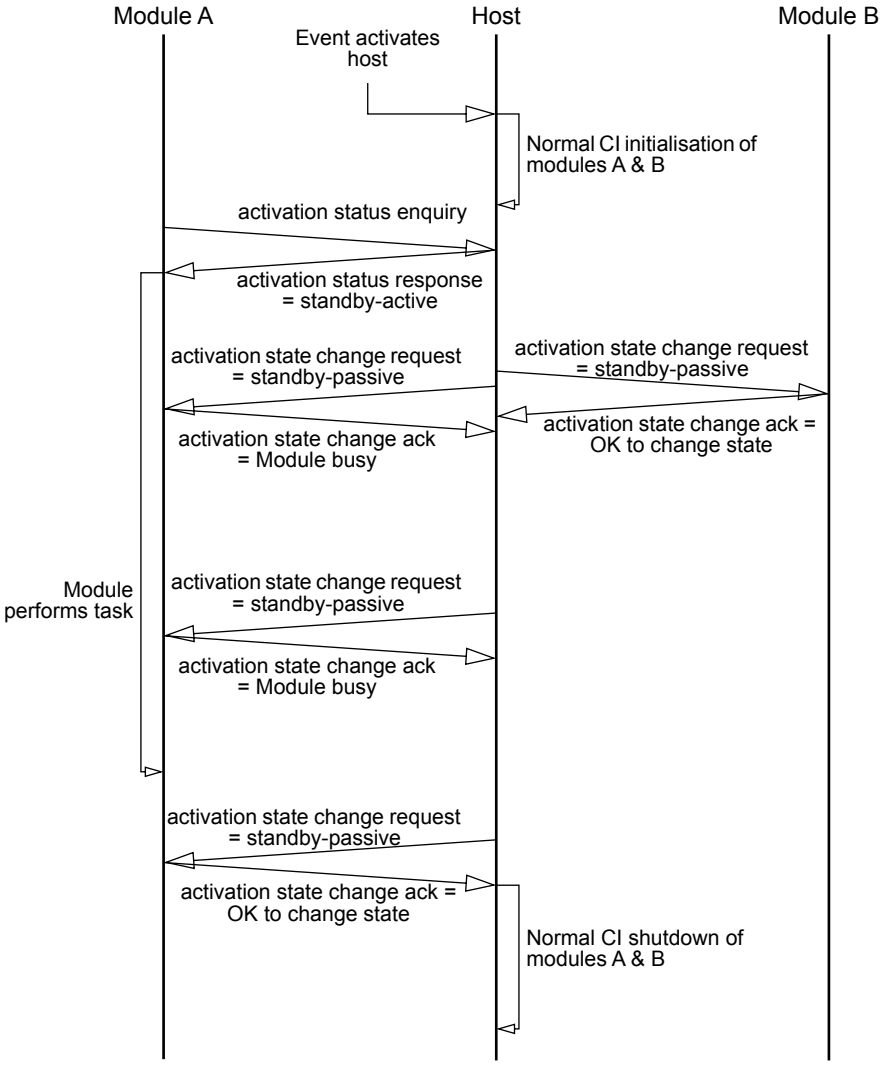

Figure 15. Module to host dialogues illustrated

6.4 Event Management

Event Manager

The *Event Manager Resource* allows modules to define events which should be signalled to the module. If the host is in standby-passive mode when the event is detected the activation state will be raised to standby-active and the module will be notified of the event.

6.4.1 Event Manager sessions

Module ID derived resource instances

The Event Manager defines the set of resource instances it shall present after the host has determined the set of modules (and Module IDs) present. One resource instance is created for each Module ID and with the Module ID in its resource_instance field.

Module Connection

If a module has an event pending it shall open a session to "its" instance of the Event Manager each time it is activated. I.e. the module derives the resource ID for the instance of the Event Manager dedicated to serving it using the resource_class defined for the Event Manager; resource_type =1; resource_instance = Module_ID and resource_version = 1.

Through this module specific session the Event Manager can send module specific messages (such as Event notification) to the module.

Example

An example is illustrated in Figure 16. Here 4 modules are inserted into a host. Two of these modules are CA modules, but only one requires timer events and as a consequence implements a module ID and the version 2 resource manager protocols (See "Extending use of the resource ID type field" on page 13). A third module also requires timer events, the fourth module has a module ID but doesn't require timer events. In this system 3 module IDs are consumed. The host creates 3 instances of the event manager populating the instance field of the resource ID with the 3 module IDs. The two modules that needs the services of the event manager then can open sessions to the appropriate instance of the event manager. The third instance of the event manager remains un-used.

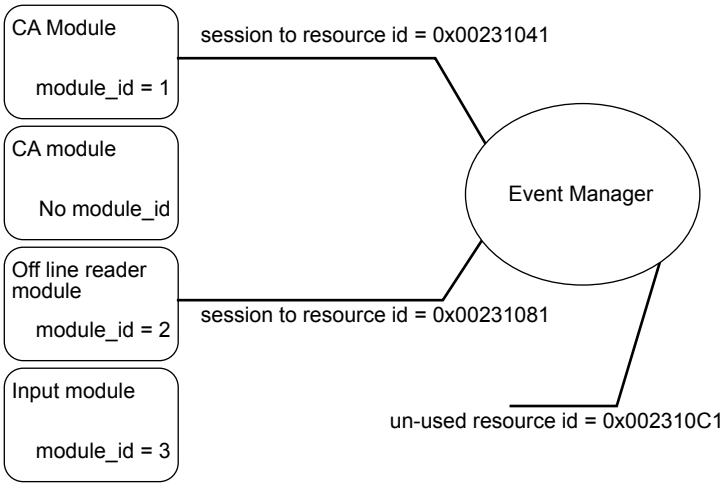

Figure 16. Use of module IDs in Event Manager resource presentation

6.4.2 Event Manager resources

Number of events

The *Event Manager* shall provide sufficient resources to retain one timer event for each transport connection provided by the host.

Retention of events
The host shall associate each timer event with the identity of the module. The scheduled event shall be retained until one of the following conditions:

- the scheduled event occurs
- the same module requests a new event (which replaces the current one)

Host's should also handle unusual conditions such as when a module reserves a timer event a long time in the future and is then removed.

6.4.3 Time range
The host shall be able to accept timer events scheduled anywhere in the future time range that can be encoded by the event request message.

6.4.4 Resource priorities
When the event is requested resource contentions at the time the event occurs cannot be predicted. The host is responsible for arbitrating the resource requirements of the module over other demands on the host. Direct[1] or indirect[2] use of the host's resources by the consumer shall have priority over demands from a module.

The system design of the module and any services associated with it are responsible for tolerating the non-availability of resources.

6.4.5 Power-up timing
The time specified by a module is the time at which it requires the host to be functioning. The host design is responsible for starting the activation process suitably before the scheduled time.

6.4.6 Energy conservation
The host Power manager may interrogate a module to determine if the host can revert to a low power consumption mode. While the module is performing the task for which it booked the timer event it may reply "Module busy, don't change state" in response to Activation state change request. When a module completes the task for which it booked the timer event it shall reply "OK to change state" when interrogated. See 6.3 on page 54.

If the user starts using the host during execution of, or shortly after completion of, the module's task then the host is responsible for determining whether to attempt to shut down the host.

6.4.7 Event request
The event request is a message sent by a module to the host to request activation of the host in response to a specified event.

Syntax	No. of bits	Mnemonic
event_request () {		
event_request_tag	24	uimsbf
length_field()		
event_type	8	bslbf
for(i=0; i<N; i++) {		
event_desc	8	bslbf
}		
}		

Table 56. Event request object

1. E.g. the consumer has directly selected a service or is interacting with a host resident application such as a programme guide.
2. E.g. the host is recording an event "booked" by the consumer via a programme guide

event_request_tag
This 24 bit field with value 0x9F8000 identifies this message.

event_type
identifier of the type of event.

event_type	description
0	Timer
1 - 255	Reserved for future use

Table 57. Coding of event types

event_desc
a block of bytes defining the event. The format of this block depends on the event type. If there are no event_desc bytes this cancels any event of this event type previously booked by this module.

event_type	Event description bytes		
		no bits	mnemonic
0	Start time (like DVB SI EIT start_time)	40	bslbf
	Duration (like DVB SI EIT duration)	24	bslbf
1 - 255	Reserved for future use		

Table 58. Coding of the event description bytes for each event type

6.4.8 Event request acknowledge

The event request reply message is sent by the host to the module in response to "Event request".

Syntax	No. of bits	Mnemonic
`event_request_ack() {`		
`  event_request_ack_tag`	24	uimsbf
`  length_field()`		
`  event_type`	8	bslbf
`  reply`	8	bslbf
`}`		

Table 59. Event

event_request_ack_tag
This 24 bit field with value 0x9F8001 identifies this message.

event_type
identifier of the type of event as described in Table 57.

reply
identifier of the type of the reply.

event_type	description
0	Event booked OK
1	Event type not supported
2	Event resources consumed
3 - 255	Reserved for future use

Table 60. Definition of event request reply codes

6.4.9 Event notification

The event notification message is sent by the host to the module when an event requested by the module occurs.

Syntax	No. of bits	Mnemonic
event_notification() {		
event_notification_tag	24	uimsbf
length_field()		
event_type	8	bslbf
}		

Table 61. Event notification object

event_notification_tag
This 24 bit field with value 0x9F8002 identifies this message.

event_type
identifier of the type of event as described in Table 57 on page 59.

6.5 Application MMI

Application MMI

The host provides an Application MMI resource with resource identifier 0x00410041. This allows a module to interact with the user by launching an application on the host's application execution environment. This is potentially much more flexible than the interaction provided by the low and high level MMIs defined in EN 50221.

The RequestStart object from the module to the host specifies the application domain required to execute the application and the reference to the initial object of the application. If the host is able to support this application domain it requests the specified initial object from the CI application using FileRequest and launches it. The module delivers this file using FileAcknowledge. Subsequent execution of the application on the host is likely to lead to further FileRequest and FileAcknowledge exchanges between the host and module as execution draws content and further executables from the module.

File Naming

The file naming convention in particular the convention for representing the parent module application as a server is application domain specific.

Application domains can specify methods for applications launched by a module onto the host to also refer to files delivered via other means (such as a broadcast stream)

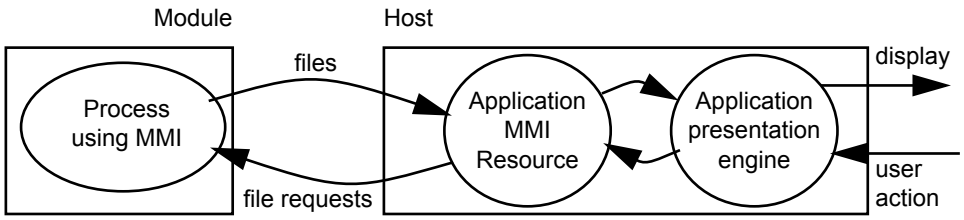

Figure 17. Overview of the operation of a Application MMI resource

6.5.1 Resource Contention

The module is not guaranteed access to the Application MMI resource. For example, if the user is interacting with a broadcast application this application has priority. Therefore there are cases (e.g. associated to a CA_PMT dialogue) where the module cannot rely on the use of this MMI method and shall be able to provide its function using another MMI method.

Cases that can be identified where a module can rely on opening a session to the Application MMI resource are:

- When responding to an EnterMenu from the host.

  (the host may need to kill an executing broadcast application - but the user focus is not on the application at the time)

- When responding to a GetServiceReq

  (as this is part of a channel change which will kill any broadcast application associated with the service selected by the user)

6.5.2 RequestStart

This message from the CI application to the host specifies both the application domain definition required by the CI application and the reference to the initial object of the application to be launched. If the host supports the requested application domain it shall in sequence:

1. Kill any application currently executing on the application execution platform requested by the RequestStart
2. Use FileRequest to request the initial object of the new application
3. Launch the new application

If the host does not support the requested application domain it shall reply with RequestStartAck.

Syntax	No. of bits	Mnemonic
RequestStart() {		
RequestStartTag	24	uimsbf
length_field()		
AppDomainIdentifierLength	8	uimsbf
InitialObjectLength	8	uimsbf
for(i=0; i<AppDomainIdentifierLength; i++) {		
AppDomainIdentifier	8	bslbf
}		
for(i=0; i<InitialObjectLength; i++) {		
InitialObject	8	bslbf
}		
}		

Table 62. RequestStart message

RequestStartTag
This 24 bit field with value 0x9F8000 identifies this message.

AppDomainIdentifierLength
This 8 bit field specifies the length of the string of bytes that specifies the application domain.

InitialObjectLength
This 8 bit field specifies the length of the string of bytes that specifies the initial object.

AppDomainIdentifier
These bytes specify the required application domain in an application domain specific way.

InitialObject
These bytes specify the initial object in an application domain specific way.

The source of the initial object may be the module (in which case FileRequest will be used to request it) or it may be another file source. The encoding of the file source within the InitialObject is a subject for application domain specification.

6.5.3 RequestStartAck

This message is sent by the host to the CI application if the requested application domain is not supported by the host.

Syntax	No. of bits	Mnemonic
RequestStartAck () {		
RequestStartAckTag	24	uimsbf
length_field()		
AckCode	8	bslbf
}		

Table 63. Request start fail message

RequestStartAckTag
This 24 bit field with value 0x9F8001 identifies this message.

AckCode
This 8 bit field communicates the response to the RequestStart:

AckCode	Meaning
0x00	Reserved for future use.
0x01	*OK* The application execution environment will attempt to load an execute the initial object specified in the RequestStart message.
0x02	*Wrong API* Application domain not supported.
0x03	*API busy* Application domain supported but not currently available.
0x04 to 0x7F	Reserved for future use.
0x80 to 0xFF	*Domain specific API busy* Application domain specific responses equivalent to response 0x03 but providing application domain specific information on why the execution environment is busy, (or not available for some other reason such as resource contention), when it will become available etc.

Table 64. AckCode values

6.5.4 FileRequest

This message from the host requests the application to deliver the named file.

Syntax	No. of bits	Mnemonic
`FileReq () {`		
`  FileReqTag`	24	uimsbf
`  length_field()`		
`  for( i=0; i<N; i++) {`		
`    FileNameByte`	8	bslbf
`  }`		
`}`		

Table 65. File request message

FileReqTag
This 24 bit field with value 0x9F8002 identifies this message.

FileNameByte
A byte of the filename requested.

6.5.5 FileAcknowledge

This message delivers the file requested by FileRequest to the host or indicates an error if the file cannot be delivered.

Syntax	No. of bits	Mnemonic
`FileAck () {`		
`  FileAckTag`	24	uimsbf
`  length_field()`		
`  Reserved`	7	bslbf
`  FileOK`	1	bslbf

Table 66. File request object

Syntax	No. of bits	Mnemonic
for(i=0; i<N; i++) {		
FileByte	8	bslbf
}		
}		

Table 66. File request object

FileAckTag
This 24 bit field with value 0x9F8003 identifies this message.

Reserved
These 7 bits are reserved for future use and shall be set to '0'

FileOK
This 1 bit field is set to '1' if the file is available and '0' otherwise.

FileByte
A byte of the file requested.

6.5.6 AppAbortRequest

This message can be sent by either host or module to request termination of the executing application process. The exact semantics are defined by the application domain. For example, the use of this call allows a process to killed without releasing the associated MMI session.

Syntax	No. of bits	Mnemonic
AppAbortReq () {		
AppAbortReqTag	24	uimsbf
length_field()		
for(i=0; i<N; i++) {		
AbortReqCode	8	bslbf
}		
}		

Table 67. File request message

AppAbortReqTag
This 24 bit field with value 0x9F8004 identifies this message.

AbortReqCode
This octet string provides and application domain specific qualification of the kill request.

6.5.7 AppAbortAck

This message is sent in response to AppAbortRequest. It allows an application domain specific response to the request for the application abort.

Syntax	No. of bits	Mnemonic
AppAbortAck () {		
AppAbortAckTag	24	uimsbf
length_field()		
for(i=0; i<N; i++) {		
AbortAckCode	8	bslbf
}		
}		

Table 68. File request object

AppAbortAckTag
This 24 bit field with value 0x9F8005 identifies this message.

AbortAckCode
This octet string provides and application domain specific response to the kill request.

6.6 Copy protection

This resource (with resource identifier 0x00041ii1) is included in hosts which support copy protection, that is, a means of controlling the content outputs from a host - audio, video and/or data - to allow or disallow recording or copying of the content. The resource provides a generic means of communicating with the copy protection function with a generalised set of objects, but the detailed content of the object will be specific to the particular copy protection system(s) implemented.

The resource consists of four objects, CP_query, CP_reply, CP_command, and CP_response. CP_query queries information and status of the resource, with the reply returned in CP_reply. CP_command sends data to the resource and CP_response sends data from the resource. The first pair of objects are specified with standard queries and replies. The second pair just pass data opaquely between application and resource, with the specific format and semantics of the data defined by the particular copy control mechanism implemented in the host.

6.6.1 Copy protection system instance management

A host may contain more than one copy protection system. For example, a host could have more than one technology to protect its output, additionally optional interfaces (for example, a digital recording interface implemented as a module) could provide copy protection features. The instance field of the resource ID (see Figure 2 on page 13) is used to differentiate each copy protection system.

6.6.1.1 Module provided systems

Where a module provides a copy protection system the module's Module ID shall be used when generating its copy protection resource ID. For example, if its Module ID is 3 then the resource ID presented is 0x000410C1.

6.6.1.2 Host provided systems

Where a host provides one or more copy protection systems it shall generate resource IDs for each system setting the instance field of the resource ID to avoid contention with any module provided systems.

6.6.1.3 Application use of copy protection systems

The set of copy protection resources apparent to an application is the combination of the host and module provided systems. An application (e.g. a CA system) that requires to control copy protection will open a session to each resource (either concurrently or sequential) and interrogate the resource to determine the system that it provides.

Applications shall only open sessions to one or more copy protection systems when its is controlling delivery of a service.

6.6.2 Copy protection system ID management

The CopyProtectionID field contains a value unique to a particular type of copy control mechanism used. This shall be a company_id allocated by the IEEE.

6.6.3 Minimum repetition interval

Copy protection systems shall not require communication between the module and the host more than once per second.

6.6.4 CP_query and CP_reply

CP_query

CP_query asks for the current status of the copy protection resource.

Syntax	No. of bits	Mnemonic
cp_query() {		
CopyProtectionQueryTag	24	uimsbf
length_field()		
CopyProtectionID	24	uimsbf
}		

Table 69. Copy protection query syntax

CopyProtectionQueryTag
This 24 bit integer with value 0x9F8000 identifies this message.

CopyProtectionID
This 24 bit value identifies the copy protection system that is to be interrogated.

CP_reply

Syntax	No. of bits	Mnemonic
cp_reply() {		
CPReplyTag	24	uimsbf
length_field()		
CopyProtectionID	24	uimsbf
Status	8	uimsbf
}		

Table 70. Copy protection reply syntax

CPReplyTag
This 24 bit integer with value 0x9F8001 identifies this message.

CopyProtectionID
As above. This field contains the true ID value for the copy protection mechanism implemented by the resource, even when the status reply is ID mismatch.

Status

status	status value
Copy Protection Inactive	01
Copy Protection Active	02
ID mismatch	FF
reserved	other values

Table 71.

6.6.5 CP_command and CP_response

These objects are identical except for the tag value.

CP_command

Syntax	No. of bits	Mnemonic
cp_command () {		
CPCommandTag	24	uimsbf
length_field()		
CopyProtectionID	24	uimsbf
for (i=0; i<n; i++) {		
CPCommandByte	8	uimsbf
}		
}		

Table 72. Copy protection command syntax

CPCommandTag
This 24 bit integer with value 0x9F8002 identifies this message.

CopyProtectionID
As defined above.

CPCommandByte
Bytes forming a command message from the application to the resource. The coding of this message is specific to the copy control technology.

CP_response

Syntax	No. of bits	Mnemonic
cp_response () {		
CPResponseTag	24	uimsbf
length_field()		
CopyProtectionID	24	uimsbf
for (i=0; i<n; i++) {		
cp_response_byte	8	uimsbf
}		
}		

Table 73. Copy protection response syntax

CPResponseTag
This 24 bit integer with value 0x9F8003 identifies this message.

CopyProtectionID
As defined above. If CP_command is sent to the resource with an invalid ID then the response is a CP_reply message with a status of ID mismatch.

CPResponseByte
Bytes forming a response message from the resource to the application. The coding of this message is specific to the copy control technology.

6.7 Software download

6.7.1 Introduction

The protocols described here provide a framework within which manufacturer specific firmware loading protocols can be implemented. They allow a CI module to be used as a source of firmware updates to a host.

6.7.2 Life cycle overview

Below is illustrated the typical life cycle of host-code download module interactions.

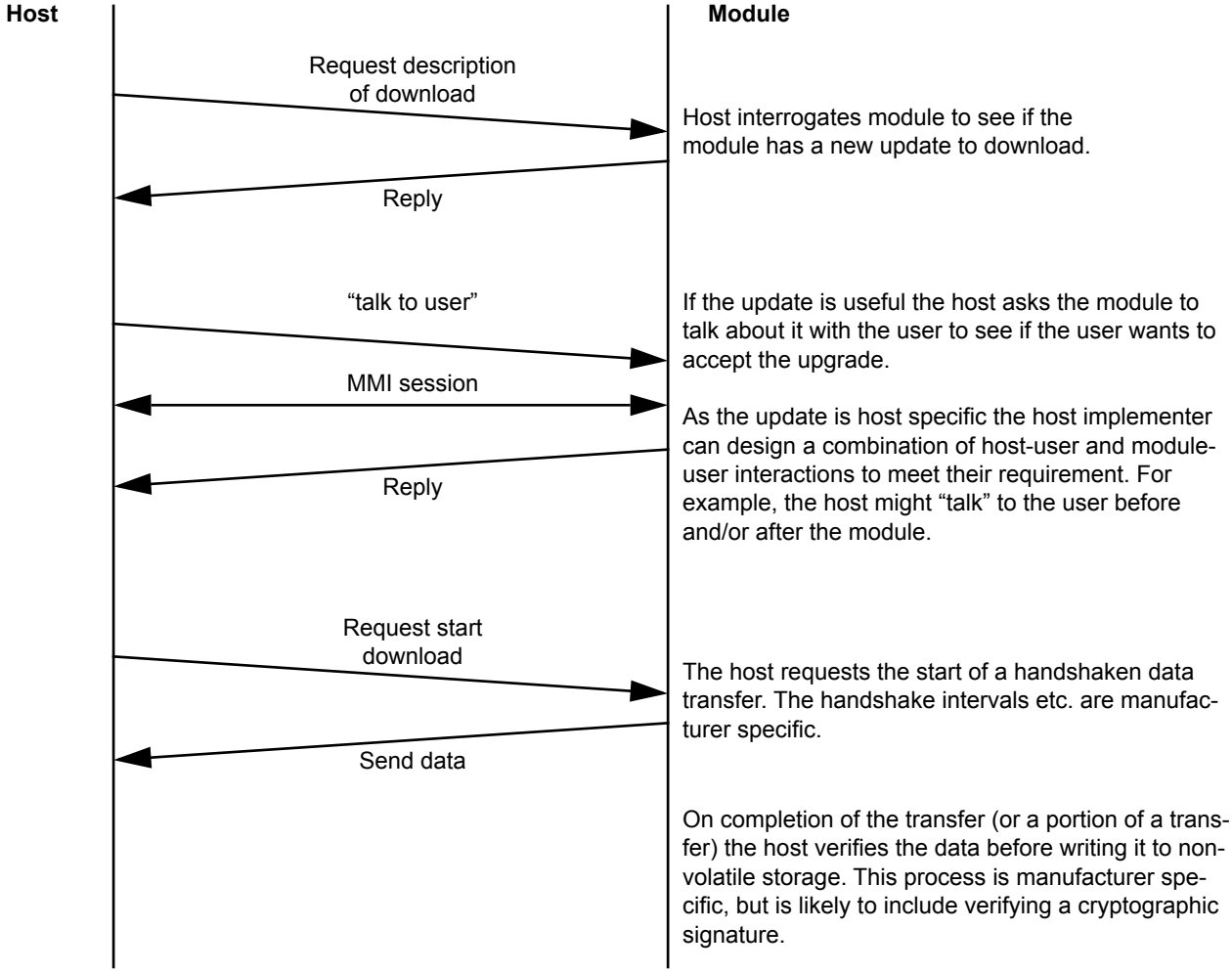

Figure 18. Life cycle overview

There are 4 distinct phases:

1. Determine if the update is required

   Unless the update provided by the module is more recent than the firmware already loaded in the host then the host can reasonably ignore the module. Host manufacturers can, at their discretion, provide methods to force the host to load an update that is already loaded or older than the one loaded. However, such actions are probably for technicians rather than users.

2. Get user authorisation

   Depending on local and manufacturer requirements a level of user authorisation will be required before the download starts. The design of this interaction is entirely in the hands of the host implementer as ultimately they are responsible for both the module and the host.

   The module contributed part of the interaction is well placed to communicate the reason for the download, for example describing the bugs fixed or features added.

3. Download the file from the module to the host

4. Verification of code before use

   The protocol described here allows transfer of data from module to host. This specification does not describe the encoding of this data. The format for conveying signatures, linking information etc. is a matter for manufacturer development.

6.7.3 Download resource

The download module provides a *Download* resource with resource ID 0x000510041. The file transfer between the module and the host is based on the DSM-CC (ISO/IEC 13818-6) User-Network Download protocol using the non-flow-controlled scenario (or optionally the flow controlled scenario). The host is the DSM-CC client and the module is the download server.

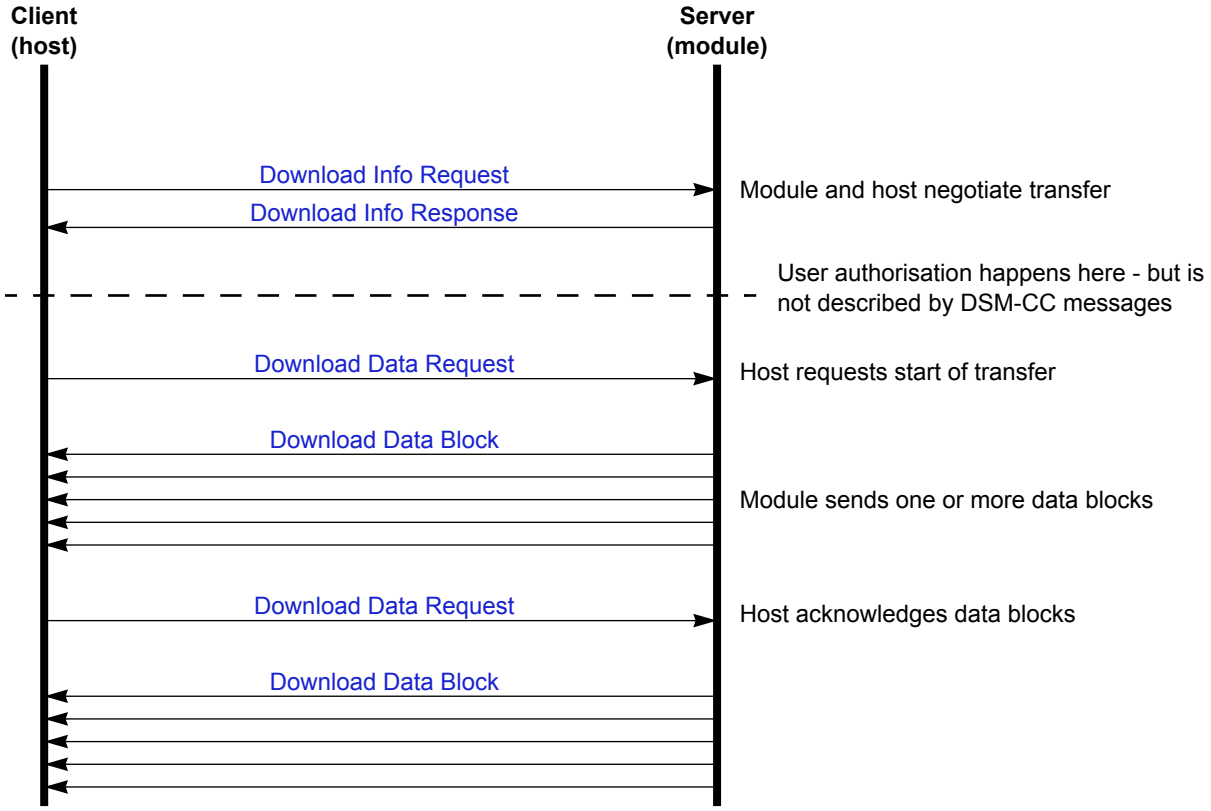

Figure 19. DSM-CC Client - Server messages

Four messages are defined for communication between the host and the module: Download Enquiry and Download Reply which encapsulate certain DSM-CC User-to-Network messages (ISO/IEC 13818-6, section 7) and User Authorisation Initiate and User Authorisation Result which help the host to determine that user authorisation has been given.

6.7.3.1 Identification of manufacturer binaries

The description of binaries is encoded on 7 bytes. In DSM-CC messages these values are carried in the corresponding fields of the compatibility descriptor:

specifier	24 bit bslbf
model	16 bit bslbf
version	16 bit bslbf

Table 74.

specifier
This 24 bit value is an IEEE OUI obtained by the manufacturer from the IEEE.

model
This 16 bit value has semantics which are specified by the organization identified by the specifier. The use of this field is intended to distinguish between various models defined by the organization.

version
This 16 bit value has semantics which are specified by the organization identified by the specifier. The use of this field is intended to distinguish between different versions of a model defined by the organization.

6.7.4 Resource-objects

6.7.4.1 Download Enquiry

Download Enquiry is used by the host to send DSM-CC messages to the module. The messages supported are:

- Download Info Request
- Download Data Request
- Download Cancel

Syntax	No. of bits	Mnemonic
`download_enq() {`		
`  download_enq_tag`	24	uimsbf
`  length_field()`		
`  for(i=0; i < n; i++) {`		
`    DSMCC_descriptor()`	8	uimsbf
`  }`		
`}`		

Table 75. Download Enquiry syntax

download_enq_tag
This 24 bit field with value 0x9F8000 identifies this message.

6.7.4.2 Download Reply

Download Reply is used by the module to send DSM-CC messages to the host. The messages supported are:

- Download Info Response
- Download Data Block
- Download Cancel

Syntax	No. of bits	Mnemonic
download_reply() {		
download_rep_tag	24	uimsbf
length_field()		
for(i=0; i < n; i++) {		
DSMCC_descriptor()	8	uimsbf
}		
}		

Table 76. Download Reply syntax

download_rep_tag
This 24 bit field with value 0x9F8001 identifies this message.

6.7.4.3 User Authorisation Initiate

User Authorisation Initiate is sent from the host to the module requesting that the module obtain user authorisation to initiate a firmware download to the host of a specified binary (see "Identification of manufacturer binaries" on page 71). The method by which the module communicates with the user is not specified. However, after sending this object the host shall enable the module to open an MMI session.

Syntax	No. of bits	Mnemonic
user_authorisation_initiate() {		
user_authorisation_initiate_tag	24	uimsbf
length_field()		
specifier	24	bslbf
model	16	bslbf
version	16	bslbf
for(i=0; i < n; i++) {		
data_byte	8	bslbf
}		

Table 77. User Authorisation Initiate syntax

user_authorisation_initiate_tag
This 24 bit field with value 0x9F8002 identifies this message.

data_byte
These data bytes are an optional field with meaning defined by the specifier.

6.7.4.4 User Authorisation Result

User Authorisation Result is sent from the module to the host. It indicates if the user has agreed to the download of the specified binary.

Syntax	No. of bits	Mnemonic
user_authorisation_result() {		
user_authorisation_result_tag	24	uimsbf
length_field()		
specifier	24	bslbf
model	16	bslbf
version	16	bslbf
for(i=0; i < n; i++) {		
result_byte	8	uimsbf
}		
}		

Table 78. User Authorisation Result syntax

user_authorisation_result_tag
This 24 bit field with value 0x9F8003 identifies this message.

result_byte
These result bytes convey the user response and have meaning defined by the specifier.

6.7.5 Host-module exchanges

The following DSM-CC messages are reproduced from ISO/IEC 13818-6.

6.7.5.1 Initial host-module negotiation

Download Info Request

When a host that supports firmware download from a module detects a module providing the download resource it opens a session to that resource and sends a Download Enquiry object encapsulating a Download Info Request message. This message communicates to the module:

- The firmware version(s) currently loaded in the host (conveyed in one or more compatibility descriptors)

  The specifier defines the meaning if more than one compatibility descriptor is conveyed.

  The specifier defines the meaning of any subdescriptor information carried within the compatibility descriptor.

- The buffer size / maximum block size that can be accommodated by the host

The Download Info Request may also carry other data defined by the specifier in the adaptation data bytes, additional information and private data bytes.

The transactionID is a value assigned by the client (host), in accordance with DSM-CC the 2 most significant bits must be set to zero, the other bits are determined by the client.

Syntax	DSM-CC element	Size (bytes)	Value	
DownloadInfoRequest() {				
protocolDiscriminator	dsmccMessage-Header	1	0x11	MPEG-2 DSM-CC
dsmccType		1	0x03	U-N Download message
messageId		2	0x1001	DownloadInfoRequest
transactionId		4		Client assigned
reserved		1	0xFF	
adaptationLength		1		
messageLength		2		
if(adaptationLength>0) {				
adaptationType	dsmccAdapta-tionHeader	1		
for(i=0; i < (adaptationLength-1); i++) {				Optional CA or private information
adaptationDataByte		1		
}				
}				
bufferSize		4		
maximumBlockSize		2		
compatibilityDescriptorLength	compatibility-Descriptor	2		
descriptorCount		2		
for (i=0; i < descriptorCount; i++) {				
descriptorType		1	0x02	System Software
descriptorLength		1		
specifierType		1	0x01	IEEE OUI
specifierData		3		OUI
model		2		
version		2		
subDescriptorCount		1		
for (j=0; j < subDescriptorCount; j++) {				
subDescriptorType	subDescriptor	1		
subDescriptorLength		1		
for (k=0; k < subDescriptorLength; k++)				
additionalInformation		1		
}				
}				
}				
privateDataLength		2		
for(i=0; i < privateDataLength; i++) {				
privateDataByte		1		
}				
}				

Table 79. Download Info Request

Download Info Response

The server (module) replies to the Download Info Request with a Download Reply object which encapsulates either a Download Info Response or a Download Cancel message. A Download Info Response communicates to the host:

- Relevant firmware version(s) that can be updated by the module (conveyed in one or more compatibility descriptors)

  The specifier defines the meaning if more than one compatibility descriptor is conveyed. If the module is not compatible with the host then zero compatibility descriptors shall be returned.

- The downloadId that shall be used to identify the download.
- The buffer size, windowSize, ackPeriod etc. that characterise the dynamics of the download.

The Download Info Response may also carry other data defined by the specifier in the adaptation data bytes, additional information, moduleInfo and private data bytes.

The transactionID matches that in the Download Info Request.

Syntax	DSM-CC element	Size (bytes)	Value	
DownloadInfoResponse() {				
protocolDiscriminator	dsmccMessage-Header	1	0x11	MPEG-2 DSM-CC
dsmccType		1	0x03	U-N Download message
messageId		2	0x1002	DownloadInfoResponse
transactionId		4		Client assigned
reserved		1	0xFF	
adaptationLength		1		
messageLength		2		
if(adaptationLength>0) {				
adaptationType	dsmccAdapta-tionHeader	1		
for(i=0; i < (adaptationLength-1); i++) {				Optional CA or private information
adaptationDataByte		1		
}				
}				
downloadId		4		
blockSize		2		
windowSize		1		
ackPeriod		1		
tCDownloadWindow		4		
tCDownloadScenario		4		
compatibilityDescriptorLength	compatibility-Descriptor	2		
descriptorCount		2		
for (i=0; i < descriptorCount; i++) {				
descriptorType		1	0x02	System Software
descriptorLength		1		
specifierType		1	0x01	IEEE OUI
specifierData		3		OUI
model		2		
version		2		
subDescriptorCount		1		
for (j=0; j < subDescriptorCount; j++) {				
subDescriptorType	subDescriptor	1		
subDescriptorLength		1		
for (k=0; k<subDescriptorLength; k++)				
additionalInformation		1		

Table 80. Download Info Response (Sheet 1 of 2)

Syntax	DSM-CC element	Size (bytes)	Value	
}				
}				
}				
numberOfModules		2		
for(i=0; i < numberOfModules; i++) {				
moduleId		2		
moduleSize		4		
moduleVersion		1		
moduleInfoLength		1		
for(i=0; i < moduleInfoLength; i++) {				
moduleInfoByte		1		
}				
}				
privateDataLength		2		
for(i=0; i < privateDataLength; i++) {				
privateDataByte		1		
}				
}				

Table 80. Download Info Response (Sheet 2 of 2)

6.7.5.2 User Authorisation

If the client (host) is satisfied that the download is technically appropriate it has the option to seek user authorisation prior to initiating the download. Seeking user authorisation may be a requirement in some markets and is in all case strongly recommended. The User Authorisation Initiate and User Authorisation Result objects described on page 72 support this.

If the client (host) determines not to proceed with the download it shall send a Download Cancel to the server (module) with the downloadCancelReason rsnAbort (or a specifier private reason).

Syntax	DSM-CC element	Size (bytes)	Value	
DownloadCancel() {				
protocolDiscriminator	dsmccMessage-Header	1	0x11	MPEG-2 DSM-CC
dsmccType		1	0x03	U-N Download message
messageId		2	0x1005	DownloadCancel
transactionId		4		Server assigned
reserved		1	0xFF	
adaptationLength		1		
messageLength		2		
if(adaptationLength>0) {				
adaptationType	dsmccAdapta-tionHeader	1		
for(i=0; i < (adaptationLength-1); i++) {				Optional CA or private information
adaptationDataByte		1		
}				
}				
downloadId		4		
moduleId		2		
blockNumber		2		
downloadCancelReason		1		
privateDataLength		2		
for(i=0; i < privateDataLength; i++) {				

Table 81. Download Cancel (Sheet 1 of 2)

Syntax	DSM-CC element	Size (bytes)	Value	
privateDataByte		1		
}				
}				

Table 81. Download Cancel (Sheet 2 of 2)

6.7.5.3 Data Download

Download Data Request

A Download Data Request message (encapsulated in a Download Enquiry object) is sent by the client (host) to the server to initiate the download using the DownloadID provided by the Download Info Response using downloadReason rsnStart. Alternatively, Download Cancel can be issued to terminate the transfer.

Subsequently further Download Data Request messages are sent to acknowledge transfer of blocks and ultimately completion of the transfer as described by DSM-CC or Download Cancel can be issued to terminate the transfer.

Syntax	DSM-CC element	Size (bytes)	Value	
DownloadDataRequest() {				
protocolDiscriminator	dsmccDownloadDataHeader	1	0x11	MPEG-2 DSM-CC
dsmccType		1	0x03	U-N Download message
messageId		2	0x1004	DownloadDataRequest
DownloadId		4		
reserved		1	0xFF	
adaptationLength		1		
messageLength		2		
if(adaptationLength>0) {				
adaptationType	dsmccAdaptationHeader	1		
for(i=0; i < (adaptationLength-1); i++) {				Optional CA or private information
adaptationDataByte		1		
}				
}				
moduleId		2		
blockNumber		2		
downloadReason		1		
}				

Table 82. Download Data Request

Download Data Block
In response to Download Data Request messages the server (module) shall transmit portions of the data to be downloaded in Download Data Block messages as described by DSM-CC.

Syntax	DSM-CC element	Size (bytes)	Value	
DownloadDataBlock() {				
protocolDiscriminator	dsmccDownloadDataHeader	1	0x11	MPEG-2 DSM-CC
dsmccType		1	0x03	U-N Download message
messageId		2	0x1003	DownloadDataBlock
DownloadId		4		
reserved		1	0xFF	
adaptationLength		1		
messageLength		2		
if(adaptationLength>0) {				
adaptationType	dsmccAdaptationHeader	1		
for(i=0; i < (adaptationLength-1); i++) {				Optional CA or private information
adaptationDataByte		1		
}				
}				
moduleId		2		
moduleVersion		1		
reserved		1		
blockNumber		2		
for(i=0; i < N; i++) {				
blockDataByte		1		
}				
}				

Table 83. Download Data Block

6.7.5.4 Private Data Fields in DSM-CC messages

dsmccAdaptationHeader
This field shall either be empty or shall carry one or more DSM-CC conditional access adaptation fields. If used to carry conditional access adaptation fields the caSystemId field shall carry a CA_system_id value registered in ETR 162.

CompatibilityDescriptor SubDescriptor
This field shall either be empty or shall carry information defined by the specifier of the enclosing CompatibilityDescriptor.

PrivateData
This field shall be empty.

6.7.5.5 Minimum compatibility

Modules
Modules providing a download resource shall tolerate insertion into hosts with which they are not compatible and not disturb such hosts. For example, if the module does not recognise the host from the one or more Download Info Request messages it should respond with Download Cancel and then close the session.

Hosts
Hosts recognising the download resource shall tolerate insertion of modules with which they are not compatible. For example, if the host does not recognise the module from the one or more Download Info Response messages it should respond with Download Cancel and then close the session.

6.8 CA pipeline resource

6.8.1 Overview

The CA-pipeline resource, with resource identifier 0x00061ii1, is a module provided resource that provides a framework that can be referenced by application domains when implementing API and CA system specific interfaces between receiver hosted applications and CA systems[1].

A set of three messages is defined. Two messages provide a transfer protocol that allows sets of bytes to be transferred between the host and the module. The encoding of these messages may be evident from the CA_System_ID identifying the CA system, or it may be negotiated in a private way within the messages. This is a subject for the application domain specification that invokes this interface. Additionally, a module to host message is defined to allow modules to send an event to an application.

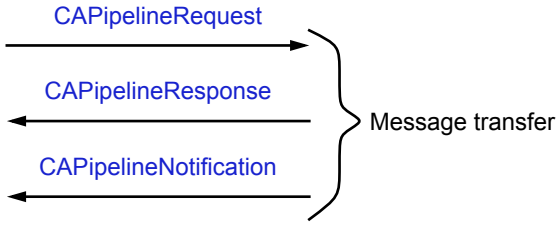

Figure 20. CA Pipeline communications

Typically the following layers will apply:

- CA system specific functions
- API specific presentation of functions
- API specific (or possibly standardised) encoding of functions into messages
- This DVB specified message transfer

6.8.2 Functionality

CA modules supporting the CA-pipeline shall present a CAP resource during profile enquiry phase. Each presents a resource ID modified by the module ID to allow multiple modules to be discriminated (see 4.1, “Extending use of the resource ID type field”, on page 13) as illustrated in Figure 21.

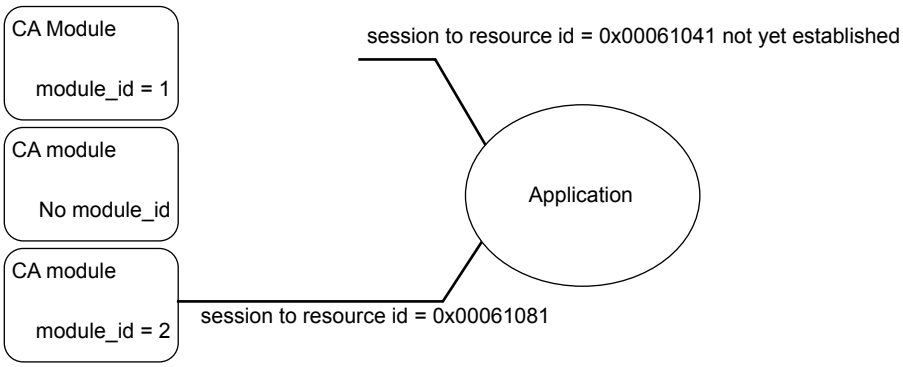

Figure 21. Use of module IDs to discriminate CAP resource instances

The API will allow applications to request a connection to specific CA system in an API specific way. The receiver is responsible for resolving this CA system connection request to the creation of CI session to the CAP resource on the appropriate module. The number of sessions supported by a CAP resource is API and CA system specific. Authors developing applications for an environment are responsible for designing within its limitations. The API & CA specification is responsible for

1. Although the specification is in terms of the common interface the same interface could be presented to applications by embedded CA systems. The details of this are outside of the scope of this specification.

providing sufficient resources or accommodating the possible failure of a connection request. Another possibility is that the API manages the multiplexing of requests from a number of requesting tasks into a single CI session. The API is responsible for presenting failure of the CI session and other errors to the application. All of these issues are outside of the scope of the CI specification.

This CA-pipeline support is located on the Module, and will offer an additional resource to the hosts resource manager during profile enquiry phase. This resource can now be used by the host to fulfil requests from the API to communicate with the CA-system in the CAM in the same way as if it communicates with an embedded CA-system. Depending on the CA-system ID requested by the API the hosts resource manager decides whether a session between the API and the CA-pipeline resource is opened or not.

6.8.3 Message Transfer

CAPipelineRequest

The CA pipeline request sends a message from the host application to the module.

Syntax	No.of bits	Mnemonic
CAPRequest() {		
CAPRequestTag	24	uimsbf
length_field()		
for (i=0; i < length; i++) {		
CASpecificData	8	uimsbf
}		
}		

Table 84. CA pipeline request syntax

CAPRequestTag
This 24 bit integer with value 0x9F8000 identifies this message.

CASpecificData
These bytes carry a CA system specific function invocation encoded in an API specific way.

CAPipelineResponse

The CA pipeline reply sends a message from the module to the application.

Syntax	No.of bits	Mnemonic
CAPResponse() {		
CAP_response_tag	24	uimsbf
length_field()		
for (i=0; i < length; i++) {		
CA_specific_Data	8	uimsbf
}		
}		

Table 85. CA pipeline response syntax

CAPRequestTag
This 24 bit integer with value 0x9F8001 identifies this message.

CASpecificData
These bytes carry the result of the CA system specific function encoded in an API specific way.

CAPipelineNotification
The CA pipeline notification sends an asynchronous message from the module to the application. Typically this is used to create an event in an API specific way. The encoding of the data in the possible CASpecificData is API specific.

Syntax	No.of bits	Mnemonic
CAPNotification() {		
CAPNotificationTag	24	uimsbf
length_field()		
for (i=0; i < length; i++) {		
CASpecificData	8	uimsbf
}		
}		

Table 86. CA pipeline event notification syntax

CAPNotificationTag
This 24 bit integer with value 0x9F8002 identifies this message.

CASpecificData
These bytes carry optional event data encoded in an API specific way.

6.8.4 Alternative implementations

This proposal does not preclude similar application to module communications using a private resource. Additionally, the use of a private resource ID as an addressing mechanism allows the interface to operate to any type of CI module, not just CA modules.

7 Definition of profiles

Three Common Interface compliance profiles are specified for receivers (hosts).

It is recommended that new hosts support at least “profile 2” rather than “profile 1”.

All receivers should conform to one of the 3 profiles and may also implement one or more domain specific extensions.

7.1 Profile 1

In “profile 1” the mandatory sections of EN 50221 (but not the annexes), with applicable amendments and corrigenda, are normative.

Support for a “low-speed communication resource class” in hosts is mandatory, but its implementation (e.g. a modem) may reside outside the host. It is not mandatory for the host to include the modem as a package in the sale of an IRD.

No object cache has to be guaranteed by the host for Low-level MMI sessions (size of the object cache can be 0).

The recommendations made in CENELEC report R206-001, “Guidelines for Implementation and Use of the Common Interface for DVB Decoder Applications” are mandatory as far as applicable to the mandatory features of EN 50221 and as far as they provide further precision to the norm EN 50221 (including errata). In case there is a contradiction with the norm EN 50221, the norm will take precedence over the guidelines.

7.2 Profile 2

In “profile 2” support for “profile 1” augmented by the following features (defined in this document) is normative:

- Resource Manager Version 2
- Application Information version 2.
- Status Query resource
- Power manager resource
- Event Management resource

During a full-screen Low-level MMI session an object cache of at least 128 kbyte shall be supported by the host.

7.3 Profile 3

In “profile 3” support for “profile 2” augmented by the following feature is normative:

- Input module support (both type-A and type-B) through a suitable application on the host, offering access to the services made available through the input module.

7.4 Domain specific extensions to profiles

A domain specific profile shall specify which base profile it is based on (1, 2 or 3) and any extensions to this profile.

These extensions can include optional features of either EN 50221 or of this specification, for example the optional features described in the annexes of EN 50221. These are fully specified and can be introduced by reference alone. The use of embedded modems may also be considered in a domain specific profile.

In addition, a domain specific profile can introduce one or more of the “frame work” features of this specification, specifically: “Copy protection”, “Application MMI” and “CA pipeline resource”. In this case, in addition to referencing the feature the domain specific profile must provide a technical specification of the implementation of the feature for that application domain.

It is envisaged that hosts supporting the DVB-MHP API will provide access to the CA pipeline resource in CA modules. It is an option for other APIs to use the CA pipeline resource for accessing the CA system in modules.

8 Resource identifiers and application object tags

Resource					Application Objects		To Resource	From Resource
Name	Identifier Value							
	class	type	vers.	identifier (hex then binary)	APDU Name	Tag Value		
ResourceManager	1	1	2	0x00010042 0000 0000 0000 0001 0000 0000 0100 0010	Profile Enquiry	0x9F8010	✓	✓
					Profile Reply	0x9F8011	✓	✓
					Profile Changed	0x9F8012	✓	✓
					Module ID Send	0x9F8013	✓	
					Module ID Command	0x9F8014		✓
ApplicationIformation	2	1	2	0x00020042 0000 0000 0000 0010 0000 0000 0100 0010	Application Info Enquiry	0x9F8020	✓	
					Application Info Info	0x9F8021		✓
					Enter Menu	0x9F8022	✓	
StreamInput	128	1*	1	0x00801ii1 0000 0000 1000 0000 0001 iiii ii00 0001	DeliverySystemInfoReq	0x9F8000	✓	
					DeliverySystemInfoAck	0x9F8001		✓
					ScanStartReq	0x9F8002	✓	
					ScanNextReq	0x9F8003	✓	
					ScanAck	0x9F8004		✓
					TuneTSReq	0x9F8005	✓	
					TuneTSAck	0x9F8006		✓

Table 87. Common interface resources (Sheet 1 of 3)

Version: DVB TM2088r3, CIT 037r11
Last Updated: 20 May 1999 16:22
Date Printed: 20 May 1999 16:23
File: ci_resources.fm

Resource					Application Objects		To Resource	From Resource
Name	Identifier Value							
	class	type	vers.	identifier (hex then binary)	APDU Name	Tag Value		
ServiceGateway (Generic Service Gateway)[a]					ServiceListReq	0x9F8000	✓	
					ServiceListAck	0x9F8001		✓
					ServiceListVersionReq	0x9F8002	✓	
					ServiceListVersionAck	0x9F8003		✓
					ServiceListChanged	0x9F8004		✓
					ServiceDescReq	0x9F8005	✓	
					ServiceDescAck	0x9F8006		✓
					GetServiceReq	0x9F8007	✓	
					GetServiceAck	0x9F8008		✓
Broadcast Service Gateway	129	1*	1	0x00811ii1 0000 0000 1000 0001 0001 iiii ii00 0001	EITSectionReq	0x9F8010	✓	
					EITSectionAck	0x9F8011		✓
Status Query	33	1*	1	0x00211ii1 0000 0000 0010 0001 0001 iiii ii00 0001	StatusQuery	0x9F8000	✓	
					Trap	0x9F8001	✓	
					GetNextItemReq	0x9F8002	✓	
					GetNextItemAck	0x9F8003		✓
					StatusAck	0x9F8004		✓
Power manager	34	1	1	0x00220041 0000 0000 0010 0010 0000 0000 0100 0001	Activation state change request	0x9F8000	✓	
					Activation state change acknowledge	0x9F8001		✓
Event Manager	35	1*	1	0x00231ii1 0000 0000 0010 0011 0001 iiii ii00 0001	Event request	0x9F8000	✓	
					Event request acknowledge	0x9F8001		✓
					Event notification	0x9F8002		✓

Table 87. Common interface resources (Sheet 2 of 3)

Resource					Application Objects		To Resource	From Resource
Name	Identifier Value							
	class	type	vers.	identifier (hex then binary)	APDU Name	Tag Value		
Application MMI	65	1	1	0x00410041 0000 0000 0100 0001 0000 0000 0100 0001	RequestStart	0x9F8000	✓	
					RequestStartAck	0x9F8001		✓
					FileRequest	0x9F8002		✓
					FileAcknowledge	0x9F8003	✓	
					AppAbortRequest	0x9F8004	✓	✓
					AppAbortAck	0x9F8005	✓	✓
Copy protection	4	1*	1	0x00041ii1 0000 0000 0000 0100 0001 iiii ii00 0001	CP_query	0x9F8000	✓	
					CP_reply	0x9F8001		✓
					CP_command	0x9F8002	✓	
					CP_response	0x9F8003		✓
Download resource	5	1	1	0x000510041 0000 0000 0000 0101 0001 0000 0100 0001	Download Enquiry	0x9F8000	✓	
					Download Reply	0x9F8001		✓
					User Authorisation Initiate	0x9F8002	✓	
					User Authorisation Result	0x9F8003		✓
CA pipeline resource	6	1	1	0x00061ii1 0000 0000 0000 0110 0001 iiii ii00 0001	CAPipelineRequest	0x9F8000	✓	
					CAPipelineResponse	0x9F8001		✓
					CAPipelineNotification	0x9F8002		✓

Table 87. Common interface resources (Sheet 3 of 3)

a] The generic service gateway is the basis for other service gateway resource. It never exists on its own. In this release the only resource based on the generic service gateway is the Broadcast service gateway.

8.1 Resource type = 1*

Where the type field of the resource ID is shown as 1* in Table 87 the 10 bit type field is 00 01ii iiii the most significant nibble of the type field indicates “type = 1” and the lower 6 bits specify a Module ID. See “Extending use of the resource ID type field” on page 13.

DVB Project Office
c/o European Broadcasting Union
17A Ancienne Route
CH-1218 Grand-Saconnex / Geneva
Switzerland
ph: +41 22 717 27 19
fax: + 41 22 717 27 27
e-mail: dvb@pax.eunet.ch