AV/C Disc Subunit General Specification

Version 1.0
January 26, 1999

Sponsored by:
Audio/Video Working Group of the 1394 Trade Association

Approved for Release by:
This document has been approved for release by the 1394 Trade Association Board of Directors

Abstract: This specification defines a model and command set for AV/C disc subunits, operating over IEEE Std. 1394-1995. The command set makes use of the Function Control Protocol (FCP) defined by IEC 61883, Digital Interface for Consumer Electronic Audio/Video Equipment, for the transport of audio/video command requests and responses. The audio/video devices are implemented as a common unit architecture within IEEE Std. 1394-1995.

Keywords:

1394 Trade Association
Regency Plaza Suite 350, 2350 Mission College Blvd., Santa Clara, CA 95054, USA
http://www.1394TA.org

1394 Trade Association Specifications are developed within Working Groups of the 1394 Trade Association, a non-profit industry association devoted to the promotion of and growth of the market for IEEE 1394-compliant products. Participants in working groups serve voluntarily and without compensation from the Trade Association. Most participants represent member organizations of the 1394 Trade Association. The specifications developed within the working groups represent a consensus of the expertise represented by the participants.

Use of a 1394 Trade Association Specification is wholly voluntary. The existence of a 1394 Trade Association Specification is not meant to imply that there are not other ways to produce, test, measure, purchase, market or provide other goods and services related to the scope of the 1394 Trade Association Specification. Furthermore, the viewpoint expressed at the time a specification is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the specification. Users are cautioned to check to determine that they have the latest revision of any 1394 Trade Association Specification.

Comments for revision of 1394 Trade Association Specifications are welcome from any interested party, regardless of membership affiliation with the 1394 Trade Association. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally, questions may arise about the meaning of specifications in relationship to specific applications. When the need for interpretations is brought to the attention of the 1394 Trade Association, the Association will initiate action to prepare appropriate responses.

Comments on specifications and requests for interpretations should be addressed to:

Editor, 1394 Trade Association
Regency Plaza Suite 350
2350 Mission College Blvd.
Santa Clara, Calif. 95054, USA

1394 Trade Association Specifications are adopted by the 1394 Trade Association without regard to patents which may exist on articles, materials or processes, or to other proprietary intellectual property which may exist within a specification. Adoption of a specification by the 1394 Trade Association does not assume any liability to any patent owner or any obligation whatsoever to those parties who rely on the specification documents. Readers of this document are advised to make an independent determination regarding the existence of intellectual property rights which may be infringed by conformance to this specification.

Table of Contents

Preface

This document is the specification for an AV/C Disc subunit. The disc is a stand-alone piece of functionality separate from a media changer mechanism or any other subunit. The disc model supports various types of disc media. The model also supports subunits which are only disc players, or which are players and recorders.

The model and data structures used for the disc are consistent with those in the AV/C Digital interface Command Set General Specification version 3.0 and the enhancements to AV/C General Specification 3.0, version1.0.

This document describes the general AV/C Disc subunit model and command set; there are separate documents for each media type specification supported by the subunit model (e.g. MiniDisc, CD…).

1. Normative References

The following documents may be useful to the reader interested in learning about the full AV/C protocol and related technologies. All standards are subject to revision; the reader is encouraged to investigate the possibility of applying the most recent editions of the documents listed below.

This AV/C Disc Model and Command Set specification must be used in conjunction with the general AV/C specification and the appropriate disc subunit media specification(s), according to the product being designed. All of these specifications are referenced below.

1.1 Contact Information

The documents referenced herein may be obtained from the following organizations:

1.1.1 1394 Trade Association (1394 TA)

The 1394 Trade Association can be contacted via the references provided on the cover page of this and all AV/C specification documents.

1.1.2 International Electrotechnical Commission (IEC) (contact in the United States)

U.S. National Committee of the IEC ANSI
11, West 42nd Street, 13th floor
New York, NY 10036

Phone:	+1-212-642-4900
	+1-212-642-4980 (sales)
Fax:	+1-212-398-0023
Internet:	http://www.ansi.org

Documents can be ordered from:

http://www.iec.ch/cs1ord-e.htm
http://www.iec.ch/cs1oi-e.htm

1.1.3 The Institute of Electrical and Electonics Engineers, Inc. (IEEE)

The IEEE can be contacted via their WWW home page: http://www.ieee.org

1.2 1394 Trade Association Specifications

[1] AV/C Digital Interface Command Set General Specificationversion 3.0 and the Enhancements to the AV/C General Specification 3.0, version 1.0.

1.3 Related Technical Specifications

[2] IEEE Std 1394-1995, *Standard for a High Performance Serial Bus*

[3] ISO/IEC 13213:1994, *Control and Status Register (CSR) Architecture for Microcomputer Buses*

1.4 AV/C Documentation Structure

The AV/C protocol supports a wide variety of media products, and it continues to grow as the protocol is refined and new media technologies are supported. As a result, it is not feasible to include all of the specification details in a single document. Therefore, the AV/C documentation suite has been organized in the following manner:

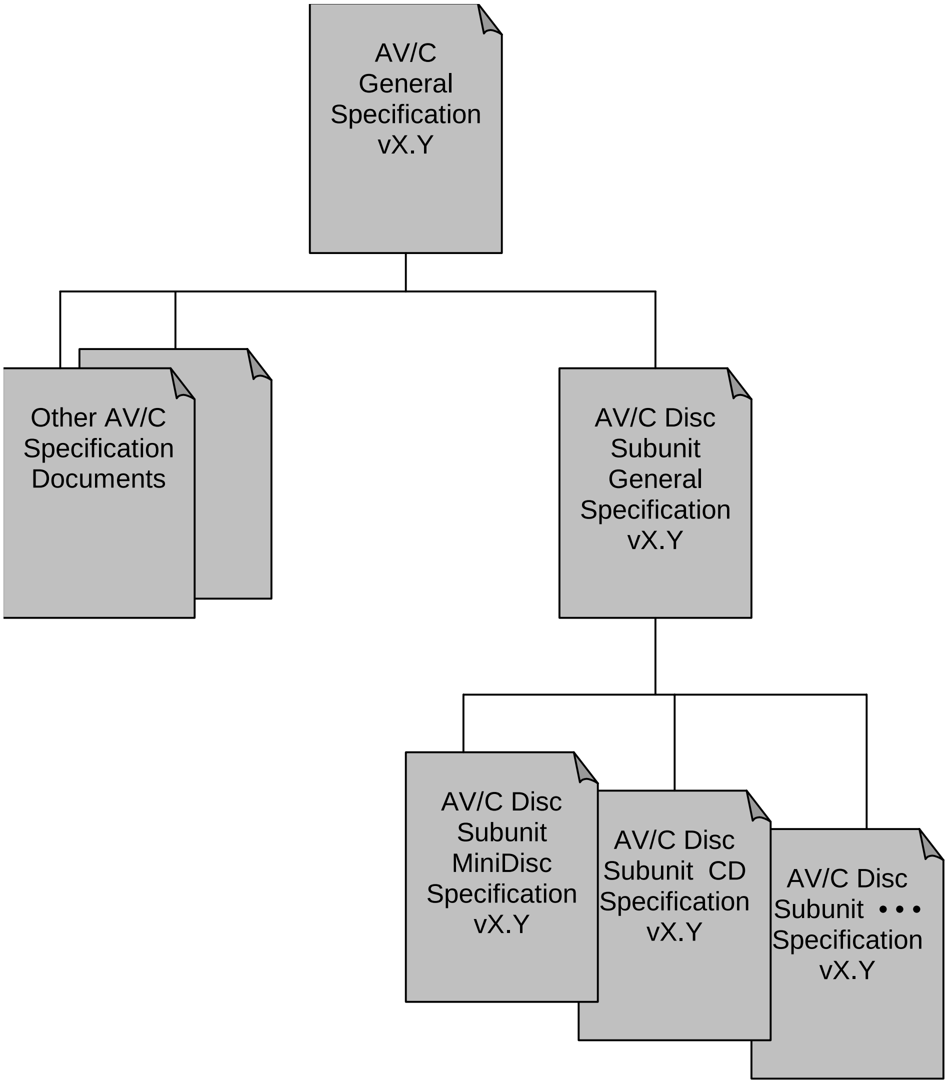

AV/C General Specification vX.Y
Other AV/C Specification Documents
AV/C Disc Subunit General Specification vX.Y
AV/C Disc Subunit MiniDisc Specification vX.Y
AV/C Disc Subunit CD Specification vX.Y
AV/C Disc Subunit • • • Specification vX.Y

Figure 1-1 AV/C document structure

At the time this is being written, there are no media-specific documents defined for the AV/C Disc Subunit model. Readers of this specification should be aware that such documents will exist in the future.

2. Changes from previous version

There are no change notes for version 1.0 of the document.

3. Definitions and abbreviations

3.1 Conformance glossary

Several keywords are used to differentiate between different levels of requirements and optionality, as follows:

expected: A keyword used to describe the behavior of the hardware or software in the design models assumed by this specification. Other hardware and software design models may also be implemented.

may: A keyword that indicates flexibility of choice with no implied preference.

shall: A keyword indicating a mandatory requirement. Designers are required to implement all such mandatory requirements to ensure interoperability with other products conforming to this specification.

should: A keyword indicating flexibility of choice with a strongly preferred alternative. Equivalent to the phrase “is recommended.”

3.2 Technical glossary

AV unit: The physical instantiation of a consumer electronic device, *e.g.*, a camcorder or a VCR, within a Serial Bus node. This document describes a command set that is part of the software unit architecture for AV units.

AV subunit: an instantiation of a virtual entity that can be identified uniquely within an AV unit and offers a set of coherent functions.

AV/C: Audio/video control, as in the AV/C Digital Interface Command Set specified by this document.

byte: Eight bits of data.

descriptor: This is a general term for a data structure which describes something, such as the subunit, individual pieces of content on media, etc.

object descriptor: An object descriptor is a data structure which describes a piece of information. This can be an audio track on a disc, a broadcast video or audio stream, etc. Object descriptors are also referred to as object entries or entry descriptors, because they are (usually) grouped into lists – see next definition.

list descriptor: A list descriptor is a data structure which describes a collection of data. It has a general header which describes the collection as a whole, then a series of object descriptors, each of which describes on piece of data in the collection. The collection of audio tracks on a compact disc can be described with a list descriptor that contains general information about the disc, and a collection of audio track object descriptors.

information block: Information blocks (also called info blocks) are enhancements to the descriptor model. Each information block contains a collection of related data fields; info blocks can also contain other info blocks.

IEEE: The Institute of Electrical and Electronics Engineers, Inc.

isochronous: A term that indicates the essential characteristic of a time-scale or signal, such that the time intervals between consecutive instances either have the same duration or durations that are integral multiples of the shortest duration. In the context of Serial Bus, “isochronous” is taken to mean a bounded worst-case latency for the transmission of data; physical and logical constraints that introduce jitter preclude the exact definition of “isochronous."

plug: A physical or virtual end-point of connection implemented by an AV unit or subunit that may receive or transmit isochronous or other data. Plugs may be Serial Bus plugs, accessible through the PCR's; they may be external, physical plugs on the AV unit; or they may be internal plug group implemented by the AV subunits.

source plug A subunit source plug is a source of output data.

destination plug A subunit destination plug receives incoming data.

Synchro Plug Group A logical grouping of several subunit source plugs. The synchro plug group is used to manage the synchronized playback of several content streams on several subunit source plugs, simultaneously.

performance: The playback of several items *sequentially* on a *single* subunit source plug.

synchronized performance: The playback of several items *simultaneously* on *several* subunit source plugs.

quadlet: Four bytes of data.

Segment A segment is part of an object. The following diagram illustrates a segmented object:

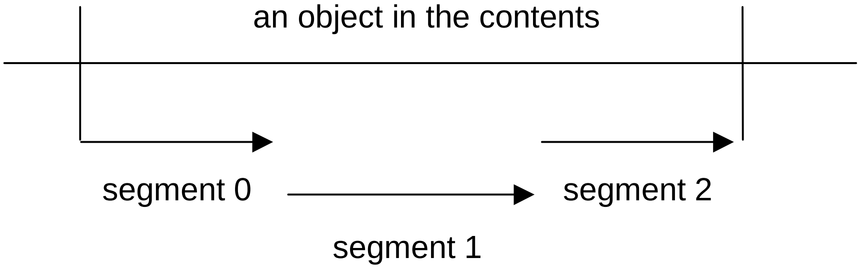


Figure 3-1 segment

Segments are useful as a means of specifying areas or locations within the scope of a single AV content object.

stream: A time-ordered set of digital data originating from one source and terminating at zero or more sinks. A stream is characterized by bounded bandwidth requirements and by synchronization points, or time stamps, within the stream data.

4. The Disc Subunit Model

4.1 The AV/C Logical Model

The following diagram illustrates the logical model of a disc subunit:

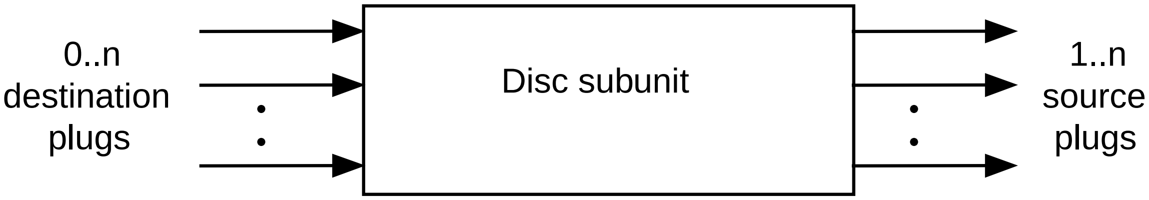


Figure 4-1 Logical model of a disc subunit

The disc subunit may have from zero to n destination plugs. If it is a non-recording disc subunit (such as a CD player), then it should have zero destination plugs. If it has the ability to record multiple simultaneous streams, then it may report n destination plugs.

There shall be at least one source plug for playing back content. If the subunit has the ability to output multiple simultaneous streams, then it may report n source plugs.

As implied in the above statements, some disc subunits may be built for playback only, while others may be built to handle both recording and playing. The subunit identifier descriptor for the disc subunit reports this capability.

4.2 AV Content and “Computer” Content

Several types of disc formats have been defined which are able to contain areas which store either traditional AV data (such as video, audio or digital still images and their descriptive information), or “computer” data (such as a typical OS file system). The AV/C disc model makes a clear distinction between these two types of storage areas, and is designed to deal only with data which is found in the AV content area. If a disc is formatted to contain two areas, one for AV and one for computer data, then all of the AV/C commands should affect only the AV content area of the media.

The following diagram illustrates this concept:

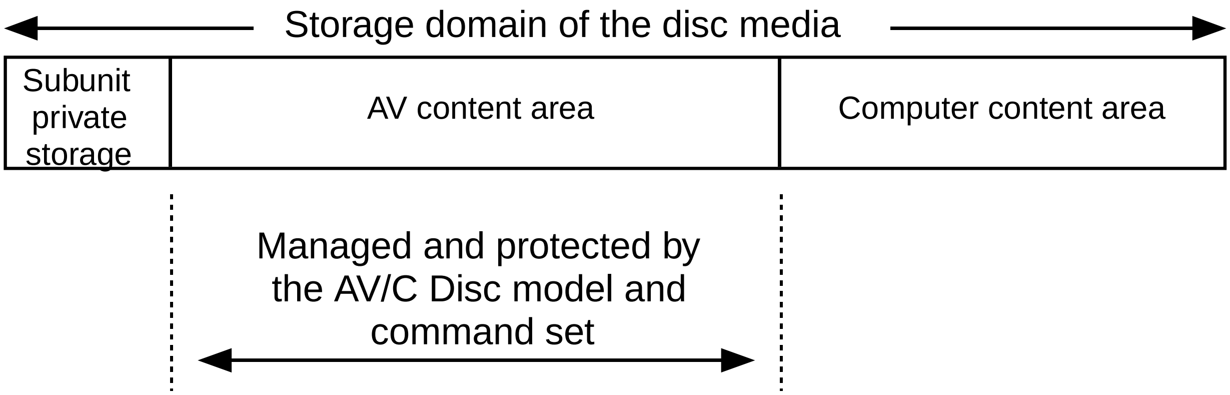

Storage domain of the disc media
Subunit private storage
AV content area
Computer content area
Managed and protected by the AV/C Disc model and command set

Figure 4-2 AV Content and “Computer” Content

The AV content area stores all of the data recorded using the AV/C commands (or content that was placed on the disc during manufacture). Included in this area is the descriptor data (objects, object lists, etc.). Controllers wishing to gain access to the computer content area should use the appropriate command set and transmission methods. The existence of the computer content area is not even visible to controllers examining the AV/C disc subunit data structures.

Note that the AV content area is a logically contiguous area, and is not partitioned into multiple AV content areas. Such partitioning is not necessary, and it not supported by the model. The logical file system described below allows a flexible storage model and content collection mechanism. Partitioning serves no useful purpose in this context.

4.3 Security of Copyrighted Content

Due to high bandwidth digital transmission and very high disc storage capacities, it is now becoming practical for consumers to record significant pieces of AV content on disc media. The AV/C data structures which are used to navigate and select content (objects and object lists, status descriptors, etc.) provide an abstraction for the underlying file system used to manage the organization of AV content on the media.

An AV/C compliant disc subunit should NOT allow controllers to gain access to any data in the AV content area (neither AV content nor descriptor data) other than by using the commands and data structures defined by the AV/C Disc Subunit specification. When these commands are invoked, the appropriate security measures should be triggered to ensure the protection of copyrighted material as it leaves the subunit and the AV/C unit in which it exists. This protection should be within the scope of the 1394 copyright solution.

4.4 Logical File System

The AV/C Disc model and command set makes use of the general AV/C objects and object list structures, as mentioned in the introduction. These data structures provide not only the storage abstraction allowing secure storage and access to protected material as described above, but they also allow a disc subunit to implement a flexible storage strategy for AV content.

The simplest storage mechanism would be a flat list of content descriptors. This would be applicable for basic disc media in which a list of audio tracks is all that’s needed to represent the content of the media. A more flexible storage model would be a hierarchical system of lists and descriptors, which allows controllers (users) to organize AV content in a manner similar to the typical computer system of directories within directories.

In the hierarchical system, objects representing audio tracks, video clips, or digital still images are arranged as members of object lists. Each object list essentially represents a directory, and the AV/C object list hierarchy model allows directories to be stored within directories to any arbitrary level.

Objects and even entire object lists may be locked in order to prevent accidental destruction. The object locking mechanism is very simple, and does not involve user authentication or any other security mechanisms.

4.5 Fragmentation and Other Storage Issues

The AV/C Disc subunit model does not expose fragmentation of the AV content area to controllers. It is the responsibility of the subunit implementation to manage the storage of AV information at all times (during recording, editing, erasing, etc.) to ensure the highest possible quality of playback performance. In this case, quality of playback essentially refers to a steady output flow of the original data, with minimal or no impact caused by fragmentation of content.

4.6 Rules for Reserved Fields

This section clarifies the rules which have always been in effect regarding how reserved fields shall be treated in command parameters and data structures.

Unless otherwise specified (see note below), command parameters and data structure fields marked as “reserved” or “reserved for future specification” shall be set to zero by controllers on input to a target, and by targets on output to controllers.

For input operands of commands, targets shall NOT ignore fields that were reserved when the target was implemented. Rather, the target shall examine the reserved fields; if any of them are set to non-reserved values, then the target shall reject the command with a NOT IMPLEMENTED response.

On output data structures or parameters of commands, controllers shall ignore fields that were reserved when the controller was implemented. These rules exist to allow future extension of the specification while retaining compatibility with existing products.

> **NOTE:** In some instances, reserved command operands or data structure fields may be specified as non-zero values. These cases will be clearly indicated in the specification. Controllers and targets shall deal with them in the same manner as defined above.

5. Disc Subunit Descriptor Structures

The AV/C Disc Subunit follows the same basic descriptor structure mechanism as defined in reference [1].

5.1 Disc Subunit-Specific Descriptor Identifiers

The general AV/C descriptor mechanism defines several ways of referenceing structures (objects, lists, etc.). These are defined in the OPEN DESCRIPTOR command in reference [1].

In addition to the general *descriptor_identifier* types, the AV/C disc subunit model defines the following reference types:

AV/C Disc Subunit-specific descriptor_identifier types	
descriptor_type	meaning
80_{16}	Disc subunit status descriptor – defined in section 7.1.5 on page 19
all others in the subunit-specific range (80_{16} - BF_{16})	reserved for future specification

Table 5-1 AV/C Disc Subunit-specific descriptor_identifier types

6. Disc Subunit Identifier Descriptor

The general subunit identifier structure is described in reference [1]. The disc subunit identifier descriptor contains the following information:

Disc Subunit Identifier Descriptor	
Address	Contents
00 00$_{16}$	descriptor_length
00 01$_{16}$	
00 02$_{16}$	generation_ID
00 03$_{16}$	size_of_list_ID
00 04$_{16}$	size_of_object_ID
00 05$_{16}$	size_of_object_position
00 06$_{16}$	number_of_root_object_lists (n)
00 07$_{16}$	
00 08$_{16}$	root_object_list_id_0
:	
:	:
:	root_object_list_id_n-1
:	
:	disc_subunit_dependent_length
:	
:	disc_subunit_dependent_information
:	
:	
:	manufacturer_dependent_length
:	
:	manufacturer_dependent_information
:	
:	

Table 6-1 Disc Subunit Identifier Descriptor

The *descriptor_length* field contains the number of bytes which follow in this descriptor structure. The value of this field does *not* include the length field itself.

The *generation_ID* field specifies which AV/C descriptor format is used by this subunit for all data structures it maintains, and the command sets which affect them. This field can have one of the following values:

generation_ID Values	
generation_ID	Meaning
00_{16}	Data structure and command sets as specified in the AV/C General specification, version 3.0
01_{16}	Data structure and command sets as specified in the AV/C General specification, version 3.0 and the Enhancements to the AV/C General Specification 3.0, version 1.0
all others	Reserved for future specification

Table 6-2 generation_ID Values

The *size_of_list_ID* field indicates the number of bytes used to indicate a list ID for this subunit. All lists maintained within the scope of this subunit shall use this number of bytes for their ID values.

The *size_of_object_ID* field specifies the number of bytes used for object_ID values managed by this disc subunit. All objects maintained within the scope of the subunit which have an ID should use this number of bytes for their ID. It is possible for some objects within the scope of a subunit to have ID values, and for some to not have ID values. If the subunit doesn't support object ID values for any of its objects, this field shall be set to 0.

The *size_of_object_position* field indicates the number of bytes used when referring to an object by its position in a list. All such reference used with the subunit shall use this number of bytes for the position reference.

The *number_of_root_object_lists* field contains the number of object lists directly associated with this subunit. This field is 2 bytes in size.

The *root_object_list_id_x* fields are the ID values for each of the associated object lists. The *number_of_root_object_lists* field indicates how many of these ID values are present.

The *disc_subunit_dependent_information* contains the following information:

Address Offset	Contents
00_{16}	disc_subunit_dependent_info_fields_length
01_{16}	
02_{16}	attributes
:	disc_subunit_version
:	number_of_supported_media_types (n)
:	supported_media_type_specification[0]
:	
:	
:	:
:	supported_media_type_specification[n - 1]
:	
:	
:	optional info blocks for future expansion
:	
:	

Table 6-3 *disc_subunit_dependent_information*

The *disc_subunit_dependent_info_fields_length* field specifies the number of bytes for the non-info block fields of the subunit dependent information; in this case, through the *supported_media_type_specification[n – 1]* structure.

Controllers should be prepared to find any number of information blocks following this field, in case the disc subunit dependent information field needs to be expanded in the future. Controllers can easily determine if any info blocks exist here by comparing the *disc_subunit_dependent_length* and *disc_subunit_dependent_info_fields_length* fields. If the following formula is true:

disc_subunit_dependent_length > (disc_subunit_dependent_info_fields_length + 2)

then info blocks exist in this structure.

The *attributes* field is defined as follows:

Attribute Value	Name	Meaning
1xxx xxxx	has_more_attributes	If this bit is set to 1, then the next byte is also an attributes byte. If this bit is 0, then the next byte is as defined for this structure.
xxxx xxx1	supports_copyright	If this bit is set to 1, then the subunit is able to honor the appropriate copyright rules when recording and playing back. If this bit is clear, then the subunit does not record copyrighted material.
all others	----------------------	Reserved for future specification.

Table 6-4 *attributes* field

The *disc_subunit_version* field indicates the version number of disc subunit command specification that this disc subunit conforms to. The upper 4 bits shows major version number, and lower 4 bits shows minor version number.

disc_subunit_version	meaning
10_{16}	Version 1.0 of the disc subunit specification
all others	Reserved for future specification

Table 6-5 disc_subunit_version

The *number_of_supported_media_types* field contains the number of different types of discs supported by this subunit. For example, an MD/CD player would support two media types.

The *supported_media_type_specification* fields are an array of *supported* (as opposed to installed) disc specifications, where each specification contains both common and type-specific entries. All supported media type specifications have this format:

Address Offset	Contents
00_{16}	supported_media_type
01_{16}	
02_{16}	implementation_profile_ID
03_{16}	media_type_attributes
:	type_dependent_length
:	
:	type_dependent_information
:	
:	

Table 6-6 *supported_media_type_specification* fields

The *supported_media_type* field identifies the type of media. The upper byte indicates the family, while the lower byte specifies more detailed specifications or about its format. The following table illustrates the defined values for the supported media type:

supported_media_type (MSB)	Value	supported_media_type (LSB)	Value
CD	01_{16}	CD-DA reserved for Video-CD	01_{16} 02_{16}
MD	03_{16}	MD-Audio reserved for Picture-MD	01_{16} 02_{16}
all others	reserved	reserved	reserved

Table 6-7 *supported_media_type* field

An example of the encoding for the supported media type would be: MD-Audio = 0301_{16}.

The *implementation_profile_ID* field specifies the profile ID of the disc implementation for this *supported_media_type*. Note that a disc subunit may be implemented with a different profile for each of the media types it supports. There shall be one profile for each supported media type. The profile definitions are described in the disc subunit media type-specific documents.

The *media_type_attributes* field is defined as follows:

Attribute Value	Name	Meaning
1xxx xxxx	has_more_attributes	If this bit is set to 1, then the next byte is also an attributes byte. If this bit is 0, then the next byte is as defined for this structure.
xxxx xxx1	can_record	If this bit is set to 1, the disc subunit has the ability to record on the specified supported media type. When this bit is clear, then this subunit is only able to play from the specified supported media type.
xxxx xx1x	supports_hierarchical_storage	If this bit is set to 1, then the subunit supports the hierarchical storage model on this media type. For more details, refer to the section titled Child Contents Lists on page 60. If this bit is clear, then the subunit supports only the flat storage model.
xxxx x1xx	supports_two_sided_media	If this bit is set to 1, then the disc subunit is able to play back two-sided media without user intervention. If this bit is set to 0, then the subunit does not support two-sided media. If the media specification does not support two sides (such as CD), this bit shall be set to 0.
all others		Reserved for future specification.

Table 6-8 *media_type_attributes* field

The *type_dependent_length* field contains the number of bytes used by the *type_dependent_information* field.

The *type_dependent_information* field contains information that is specific to each type of medium supported by the subunit. For details, please refer to the appropriate media type specification document.

The *manufacturer_dependent_length* and *manufacturer_dependent_information* fields are used for vendor-specific data. The format and contents are defined by the subunit manufacturer.

7. DISC Subunit Status Descriptor

The disc subunit status descriptor is specific to the disc subunit type. It holds information about the subunit in general, and about each of the source, destination and synchro plug group. In contrast to the subunit identifier descriptor, the information in this structure is very dynamic and is kept up to date by the subunit. This structure may be examined by a controller in order to determine the operational status of the disc and its plugs. The controller may also ask for notification of changes to this descriptor. For details, please refer to the DISC STATUS command.

7.1 The Status Reporting Model

The following sections describe the mechanism of status reporting for the AV/C disc subunit, including rules and guidelines for both targets and controllers.

7.1.1 The Size of Status Data Structures

The disc subunit status descriptor is designed to be a fixed length for a given subunit implementation (depending on the number of source, destination and synchro plug group and the type of AV objects which are supported). The purpose is to make the operation of reporting status and checking status as reliable as possible under what may be very dynamic situations. The reason for this is that controllers may need to read the status descriptor in several small chunks (due to their particular limitations in the number of bytes they can read in a single operation). If the status structures are “variable” length, then a change in status for the subunit (even on a plug that the controller does not care about) causes a change in the overall size of the status data structure. This in turn can cause a shift in position of the data which the controller does care about, and may require the controller to throw away the data it has read and go back to the beginning of its reading procedure before it has a chance to complete a check of the status.

The status descriptor is composed of a hierarchy of information blocks. At the top level is one info block for the general subunit status and one for each of the collection of destination plugs, collection of source plugs, and collection of synchro plug group. Each of the areas within the status descriptor has a fixed length. This length depends on the type of AV object which is currently being recorded or played on a destination or source plug. For example, disc subunits that deal with only one type of audio object (such as CD or MD) are able to have a fixed-length status descriptor whose size never changes.

Subunits which are able to record or play any type of AV object, such as a general storage subunit, can have a fixed length status descriptor for a given period of time. However, that status descriptor is likely to change from time to time, depending on the type of AV object(s) which are being recorded/played back on the plugs. Controllers should take this into account when reading the structures of such subunits.

Because each of the areas of the status descriptor are fixed in length, they have been designed to hold the largest number of bytes needed for one of the status conditions. This means that some of the status conditions have unused bytes in the data structure. Those bytes which are not needed by a particular reporting status shall be set to FF_{16} by the subunit when it creates and updates this structure.

7.1.2 Full Status Descriptor Access

Controllers are required to issue an OPEN DESCRIPTOR command before attempting to send a READ DESCRIPTOR command. This access mechanism helps to ensure that the controller is reading valid status information. As long as the status descriptor does not change, then it can remain open so that controllers can perform subsequent READ DESCRIPTOR commands.

Note that controllers should not keep any descriptor open for long periods of time without using it, so that other controllers have a chance to open and read it also. This is a general descriptor access etiquette rule, and is not limited only to the disc subunit status descriptpor.

When the status of *any* area does change, the subunit should force the descriptor to be closed for all controllers who had access. When a controller attempts to perform another READ DESCRIPTOR command, it is REJECTED, signaling the controller that the status has changed. The controller is needed to re-open the descriptor and examine it to determine what has changed.

Please see section 7.1.4 for details about exceptions to the status descriptor access rules.

7.1.3 Selective Status Descriptor Area Access

The disc subunit identifier descriptor model allows selective, or partial, access to certain areas of the descriptor structure. This has some benefits for controllers, because it allows them to monitor a certain set of values (such as the counter of the data stream) without worrying about other status changes.

For selective access, controllers are still required to gain access by using the OPEN DESCRIPTOR command. However, it is possible to specify a selected info block instead of the entire descriptor structure. Once accessed, the info block and all of its nested info blocks are then available to the controller.

If other changes in the status structure do occur, and they have no impact on the scope of the info block being monitored, then no interruption of monitoring is necessary. Thus, the subunit does not have to close the descriptor for the controller, and the controller does not have to re-establish access. However, if the specified area does change, then the normal procedures, as described, are followed.

Please see section 7.1.4 for details about exceptions to the status descriptor access rules.

More details on selective descriptor area access can be found in the section titled Disc Subunit Status Descriptor Identifier on page 19.

7.1.4 Updating the Status Information

The subunit should update the status descriptor when it is first opened by a controller. The scope of the update would depend on the scope of access (the entire descriptor or a specified info block and any info blocks that are nested inside of it).

On each subsequent READ DESCRIPTOR command, the subunit may choose to only update that portion which the controller is reading. This would be achieved by the subunit examining the read request to determine which data is being accessed, and updating only those fields of the disc subunit status descriptor.

Exception to the descriptor access rules noted above: some status descriptors have information that is changing on a continuous basis, such as a location counter. During play or record operations, these values are changing so frequently that it would be very inefficient for the overall AV network if the subunit was constantly closing the descriptor at each counter update, and thus forcing controllers to re-open and re-read the descriptors. To improve system performance, subunits are NOT required to close the status descriptors when such continuously changing fields are being updated.

Controllers are strongly recommended to read such continuously changing data as a "whole", rather than in parts, to ensure that the collection of data is valid. For example, if a controller reads the hours, minutes and seconds fields of a counter in one read operation, and then the frames field in another, then its display of hours:minutes:seconds:frames may be incorrect because the frame count may have rolled over, causing a change in the seconds count, between the two read operations.

The info blocks which are considered to be frequently updated are noted below (this table will be expanded in the future when additional frequently-changing data is defined):

frequently updated info blocks	
info block type	name
00 03_{16}	position_info_block (a general info block, not disc subunit-specific)
88 09_{16}	audio_level_meter_status_info_block

Table 7-1 frequently updated info blocks

Note that media capacity information is found in the contents list header, because capacity refers to the entire media. This is not considered a "frequently changing" field because the implementation can update this information at any appropriate time (it's not necessary to continuously update it). The current position information refers to a given stream of data on a source plug, so it is found in the subunit status descriptor.

7.1.5 Disc Subunit Status Descriptor Identifier

The disc status descriptor is specific to the disc subunit type; it has the following type value from the range of subunit-specific descriptor types (see also section 5.1 Disc Subunit-Specific Descriptor Identifiers on page 11):

descriptor_type	Meaning
80_{16}	Disc Status Descriptor

Table 7-2 descriptor_type

The following diagram illustrates the general AV/C *descriptor_identifier* structure format (explained in detail in reference [1]):

general AV/C descriptor_identifier format	
address_offset	Meaning
00_{16}	descriptor_type (= 80_{16} for disc status descriptor)
01_{16}	descriptor_type_specific_reference
:	
:	

Table 7-3 general AV/C descriptor_identifier format

The *descriptor_type_specific_reference* field contains a detailed specification for how to refer to the disc subunit status descriptor. As described in previous sections, it is possible to access the full status descriptor or only specified parts of it. This is accomplished by referring to the descriptor in a certain way. The following diagram illustrates the format of the *descriptor_type_specific_reference* field:

disc subunit status descriptor descriptor_type_specific_reference	
address_offset	Meaning
00_{16}	reference_method
01_{16}	reference_method_specific
:	
:	

Table 7-4 disc subunit status descriptor descriptor_type_specific_reference

The *reference_method* field indicates how the status descriptor is being referenced. The following methods are defined:

reference_method	meaning
00_{16}	full_descriptor_reference
01_{16}	info_block_reference
all others	Reserved for future specification

Table 7-5 reference_method

7.1.5.1 Full Descriptor Reference

address_offset	Meaning
00_{16}	Disc Status Descriptor descriptor_type = 80_{16}
01_{16}	reference_method = 00_{16} (full_descriptor_reference)

Table 7-6 Full Descriptor Reference

When the *full_descriptor_reference* method is used, then the entire disc subunit status descriptor is being referenced. There is no *reference_method_specific* information for this method.

7.1.5.2 Information Block Reference

The information block reference method allows the controller to specify any information block nested in the status descriptor structure:

address_offset	Meaning
00_{16}	Disc Status Descriptor descriptor_type = 80_{16}
01_{16}	01_{16} (information block reference)
02_{16}	info_block_reference_path
:	
:	

Table 7-7 information block reference

The *info_block_reference_path* field is the same as described in reference[1]. When used in this situation (referring to an info block in disc status descriptor), it still contains a full path reference, beginning with the status descriptor reference type, down through the target information block.

7.1.6 Disc Subunit Status Descriptor Access Requirements

The disc subunit is required to allow controllers to access the full status descriptor (reference_method = 00_{16}). Supporting the info block access methods is optional. If info block access is supported, then the scope of nesting access supported by the subunit is optional. For example, if controllers want to access the status information for a source plug, the subunit may require them to access the entire source plug status area information block, and might not support access to individual source plug status info blocks.

Also, the number of controllers which can be supported at the same time is up to the implementation.

7.2 The Disc Subunit Status Descriptor

The general format of the disc subunit status descriptor is as follows:

Disc Subunit Status Descriptor	
Address	Contents
$00\ 00_{16}$	descriptor_length
$00\ 01_{16}$	
$00\ 02_{16}$	general_disc_subunit_status_info_block
:	
:	
:	destination_plug_status_area_info_block
:	
:	
:	source_plug_status_area_info_block
:	
:	
:	synchro_plug_group_status_area_info_block
:	

Table 7-8 Disc Subunit Status Descriptor

The *descriptor_length* field specifies the number of bytes for the remainder of the status descriptor structure, not including the *descriptor_length* field.

Each of the main information blocks shown contains several other info blocks. They are described in detail in the following sections.

Controllers should be prepared for any nesting level of information blocks, and should be prepared to find other types of info blocks at any time. This structure may be extended with additional info blocks in the future.

7.2.1 General Disc Subunit Status Area Info Block (88 00$_{16}$)

The *general_disc_subunit_status_area_info_block* contains status information about the disc subunit which is not specific to a particular destination or source plug. It has the following format:

general_disc_subunit_status_area_info_block	
Address Offset	Contents
00 00$_{16}$	compound_length
00 01$_{16}$	
00 02$_{16}$	info_block_type = 8800$_{16}$ (general_disc_subunit_status_area_info_block)
00 03$_{16}$	
00 04$_{16}$	primary_fields_length
00 05$_{16}$	
00 06$_{16}$	media_and_edit_status_info_block
:	
:	

Table 7-9 general_disc_subunit_status_area_info_block

The *compound_length* field specifies the number of bytes for the remainder of this information block (including any nested information blocks which may occur after the last well-defined field). Note that there is at least one nested information block shown for this structure, but controllers should be prepared for others to be found as well.

The *primary_fields_length* specifies the number of bytes for the remaining well-defined fields of this structure (note that there are no well-defined fields for this info block). Any nested info blocks would appear after this.

The *media_and_edit_status_info_block* is required to be contained in the *general_disc_subunit_status_info_block* structure; additional information blocks MIGHT be found here, so controllers should not treat other info blocks as an error condition.

7.2.2 Media and Edit Status Information Block (88 04$_{16}$)

The media_and_edit_status_info_block describes the state of the disc media and any in-progress editing that might be in effect. This info block has the following format:

Address Offset	msb							lsb
00 00_{16}	compound_length							
00 01_{16}								
00 02_{16}	info_block_type = 88 04_{16} (media_and_edit_status *info*block)							
00 03_{16}								
00 04_{16}	primary_fields_length							
00 05_{16}								
00 06_{16}	disc_in_drive		error_condition		reserved			
00 07_{16}	undo_status							
00 08_{16}	difference	auto_ update	reserved					

Table 7-10 media_and_edit_status_info_block format

The *compound_length* field specifies the number of bytes for the remainder of this information block (including any nested information blocks which may occur after the last non-info block field). Note that there are no nested information blocks shown for this structure, but controllers should be prepared for any blocks to be found in case this structure is extended in the future.

The *info_block_type* field indicates that this block is for media and edit status information.

The *primary_fields_length* specifies the number of bytes for the remaining non-info block fields of this structure (through address offset 00 08_{16} in this case). Any nested info blocks would appear after this.

The *disc_in_drive* bits indicate whether a disc is inserted in the drive, as indicated below:

disc_in_drive bits	Meaning
00b	Unknown – The subunit is unable to determine if a disc is in the drive or not.
01b	Installed – There is a disc in the drive.
10b	Not installed – There is no disc in the drive.
11b	reserved for future specification

Table 7-11 disc_in_drive bits

The *error_condition* field specifies an error condition, if any, for the drive subunit:

error_contition	Meaning
00b	No error.
01b	Media error – disc inserted upside down, the TOC is unreadable, etc.
10b	Drive error – caused by the drive subunit.
11b	Error is caused by the disc or drive (cannot determine).

Table 7-12 error_contition

The *undo_status* field indicates the opcode of the AV/C command that will be undone when the subunit receives an UNDO command. If there is no command to be undone, or undo is not supported, this field shall contain FF_{16}.

The depth of the undo stack is a subunit implementation choice.

The *difference* bit indicates that there is a difference between actual contents on the disc and the temporary contents list hierarchy. This occurs when editing actions have been performed, but have not yet been updated to the media. If this bit is set to 1, there is a difference. If this bit is set to 0, there is no difference (for details, please refer to section 9.5 Temporary Contents Lists and Objects on page 62). If temporary contents lists are not supported, then editing changes are always reflected on the media; therefore, this bit shall be set to 0.

The *auto_update* bit indicates whether automatic updates between the temporary contents list hierarchy and the disc is on or off. If this bit is set to 1, automatic updates are on. If this bit is set to 0, automatic updates are off (for more details, see section 10.1 on page 83). If temporary contents lists are not supported, then this bit shall be set to 1 (updates always occur).

7.2.3 Destination Plug Status Area Info Block (88 01_{16})

The *destination_plug_status_area_info_block* contains information about the entire set of destination plugs. It has the following format:

destination_plug_status_area_info_block	
Address Offset	Contents
00 00_{16}	compound_length
00 01_{16}	
00 02_{16}	info_block_type = 88 01_{16} (destination_plug_status_area_info_block)
00 03_{16}	
00 04_{16}	primary_fields_length
00 05_{16}	
00 06_{16}	number_of_destination_plugs
00 07_{16}	nested plug_status_info_block structures
:	
:	

Table 7-13 destination_plug_status_area_info_block

The *number_of_destination_plugs* field specifies the number of destination plugs on the disc subunit, and hence it indicates the number of *plug_status_info_block* structures that are nested in this info block. The structures are located sequentially, not nested inside of each other.

The *plug_status_info_block* structures each describe the status of a plug. The type of plug they describe depends on where the structures are located. The info blocks that are found in the *destination_plug_status_area_info_block* each describe a destination plug.

The *plug_status_info_block* structures are defined in section 7.2.6 on page 26.

Controllers should be prepared to find other types of info blocks nested in this structure, and to not treat this as an error.

7.2.4 Source Plug Status Area Info Block (88 02_{16})

The *source_plug_status_area_info_block* contains information about the entire set of destination plugs. It has the following format:

source_plug_status_area_info_block	
Address Offset	Contents
00 00_{16}	compound_length
00 01_{16}	
00 02_{16}	info_block_type = 88 02_{16} (source_plug_status_area_info_block)
00 03_{16}	
00 04_{16}	primary_fields_length
00 05_{16}	
00 06_{16}	number_of_source_plugs
00 07_{16}	nested plug_status_info_block structures
:	
:	

Table 7-14 source_plug_status_area_info_block

The *number_of_source_plugs* field specifies the number of source plugs on the disc subunit, and hence it indicates the number of *plug_status_info_block* structures that are nested in this info block. The structures are located sequentially, not nested inside of each other.

The *plug_status_info_block* structures each describe the status of a plug. The type of plug they describe depends on where the structures are located. The info blocks that are found in the *source_plug_status_area_info_block* each describe a source plug.

The *plug_status_info_block* structures are defined in section 7.2.6 on page 26.

Controllers should be prepared to find other types of info blocks nested in this structure, and to not treat this as an error.

7.2.5 Synchro Plug Group Status Area Info Block (88 03_{16})

The *synchro_plug_groiup_status_area_info_block* contains information about the entire set of synchro plug group supported by the subunit. For details on the use of synchro plug group, refer to section 9.7.2. This info block has the following format:

synchro_plug_group_status_area_info_block	
Address Offset	Contents
00 00$_{16}$	compound_length
00 01$_{16}$	
00 02$_{16}$	info_block_type = 88 03$_{16}$ (synchro_plug_group_status_area_info_block)
00 03$_{16}$	
00 04$_{16}$	primary_fields_length
00 05$_{16}$	
00 06$_{16}$	number_of_synchro_plug_group
00 07$_{16}$	nested plug_status_info_block structures
:	
:	

Table 7-15 synchro_plug_group_status_area_info_block

The *number_of_synchro_plug_group* field specifies the number of synchro plug group supported by the disc subunit, and hence it indicates the number of *plug_status_info_block* structures that are nested in this info block. The structures are located sequentially, not nested inside of each other.

7.2.6 Plug Status Info Block (88 05$_{16}$)

The *plug_status_info_block[x]* info blocks each contain status information for their respective destination, source or synchro plug group. There shall be one of these structures for each plug on the disc subunit. The format is as follows:

plug_status_info_block	
Address Offset	Contents
00 00$_{16}$	compound_length
00 01$_{16}$	
00 02$_{16}$	info_block_type = 88 05$_{16}$ (plug_status_info_block)
00 03$_{16}$	
00 04$_{16}$	primary_fields_length
00 05$_{16}$	
00 06$_{16}$	plug_number
00 07$_{16}$	secondary_fields
:	
:	

Table 7-16 plug_status_info_block

The *compound_length* field specifies the number of bytes for the remainder of this information block (including any nested information blocks that may occur after the last non-info block field). Note that there are some nested information blocks shown for this structure, but controllers should be prepared for others to be found as well.

The *primary_fields_length* specifies the number of bytes for the remaining non-info block fields of this structure (through address offset 00 06$_{16}$ in this case). All nested info blocks shall appear after this.

The *plug_number* field specifies which subunit plug this structure represents. The type of plug is determined by the hierarchy in which this structure is found; if it's in the *destination_plug_status_area_info_block*, then this represents a destination plug. The same rule holds for source plugs and synchro plug group.

Following the *plug_number* field are the info blocks which are currently defined to be nested in the *plug_status_info_block*. The disc subunit profile definitions (defined in the media type-specific documents), and implementation choice, will affect how much information each of these info blocks contains. Controllers should be prepared for any length and any combination of info blocks when parsing all structures.

The following info block types are currently defined as valid for nesting inside of plug status info blocks. Controllers should be designed to allow other info block types to be found, and to not treat them as an error condition.

currently-defined info blocks to be nested in the plug_status_info_block				
Info Block Type	Info Block Name	source plug status	destination plug status	synchro plug group status
88 06$_{16}$	operating_mode_info_block	yes	yes	yes
00 03$_{16}$	position_info_block	yes	yes	yes
88 07$_{16}$	plug_configuration_info_block	yes	yes	no
88 08$_{16}$	playback_order_configuration_info_block	yes	no	yes
88 09$_{16}$	audio_level_meter_status_info_block	yes	no	no
88 0A$_{16}$	monitor_status_info_block	yes	no	no
88 0B$_{16}$	synchro_plug_group_configuration_info_block	no	no	yes

Table 7-17 currently-defined info blocks to be nested in the plug_status_info_block

Note that each of the info bocks in the table may be defined to include other info blocks.

7.2.7 Operating Mode Info Block (88 06$_{16}$)

The *operating_mode_info_block* indicates the current operating mode for a plug. This info block has a general format; whether it applies to a source, destination or a synchro plug group depends on the kind of structure in which it is contained (e.g. destination plug status). It has the following format:

operating_mode_info_block	
Address Offset	Contents
00 00$_{16}$	compound_length
00 01$_{16}$	
00 02$_{16}$	info_block_type =88 06$_{16}$ (operating_mode_info_block)
00 03$_{16}$	
00 04$_{16}$	primary_fields_length
00 05$_{16}$	
00 06$_{16}$	operating_mode
00 07$_{16}$	operating_mode_specific_information
:	
:	

Table 7-18 operating_mode_info_block

The *compound_length* field specifies the number of bytes for the remainder of this information block (including any nested information blocks which may occur after the last non-info block field). Note that there are no nested information blocks shown for this structure, but controllers should be prepared for them to be found while parsing.

The *primary_fields_length* specifies the number of bytes for the remaining non-info block fields of this structure (through the *operating_mode_specific_information* field in this case). All nested info blocks shall appear after this.

> **NOTE:** IMPORTANT: To make the monitoring of status information simpler and more reliable for controllers, a subunit implementation shall make sure that all *operating_mode_info_block* structures it chooses to support are of the same length. This length can be different among subunit implementations, but within a given implementation, the length must be consistent. This is achieved by padding the *operating_mode_info_blocks* with enough bytes to make them all match the largest one, in size. To allow for future enhancements to the status descriptor model, controllers should still be implemented to prepare for changes in the size of ANY structure, including these structures.

The *operating_mode* field specifies which mode this plug is in. The mode generally corresponds to the AV/C command opcode which caused the mode; in some cases, the mode may not be the result of a command. Some of these modes make sense only for source and synchro plug group, others only for destination plugs, and others for all types of plugs. This field may take on one of the following values:

operating_mode	Meaning
0D$_{16}$	OBJECT NUMBER SELECT – An object is being transmitted on the source plug.
50$_{16}$	SEARCH – The subunit is performing a search on a track which is on this plug.
C2$_{16}$	RECORD – The incoming stream is being recorded.
C3$_{16}$	PLAY – The plug is playing an AV object.
C5$_{16}$	STOP – The stream on the plug is currently stopped.
56$_{16}$	RECORD OBJECT – One or more incoming files (non-streaming data) are being recorded.
C7$_{16}$	REHEARSAL – The plug is in rehearsal mode.
FF$_{16}$	SUSPENDED – The plug is currently unavailable for some reason. The reason is specified in the specific info fields.
all other values	Reserved for future specification.

Table 7-19 *operating_mode* field

The *operating_mode_specific_information* field contains values which are specific to each of the operating modes as defined above. The following values are defined:

Address Offset	msb							lsb
	operating_mode_specific_information for mode OBJECT NUMBER SELECT (0D$_{16}$)							
00$_{16}$	<< no additional information – FF$_{16}$ pad bytes if necessary>>							
:	:							

Table 7-20 *operating_mode_specific_information* field

As shown, there are no additional values defined for the *operating_mode_specific_information* when the mode is OBJECT NUMBER SELECT.

Address Offset	msb							lsb
operating_mode_specific_information for mode SEARCH (50_{16})								
00_{16}	search_type							
:	<<FF$_{16}$ pad bytes if necessary>>							

Table 7-21 operating_mode_specific_information for mode SEARCH (50_{16})

The fields *search_type* is as defined for the SEARCH command. Please refer to the description of that command for details.

Address Offset	msb							lsb
operating_mode_specific_information for mode RECORD ($C2_{16}$)								
00_{16}	subfunction_1							
01_{16}	subfunction_2							
:	<<FF$_{16}$ pad bytes if necessary>>							

Table 7-22 operating_mode_specific_information for mode RECORD ($C2_{16}$)

The fields *subfunction_1* and *subfunction_2* are as defined for the RECORD command. Please refer to the description of that command for details.

Address Offset	msb							lsb
operating_mode_specific_information for mode PLAY ($C3_{16}$)								
00_{16}	subfunction_1							
:	<<FF$_{16}$ pad bytes if necessary>>							

Table 7-23 operating_mode_specific_information for mode PLAY ($C3_{16}$)

The *subfunction_1* field is as defined for the PLAY command. Please refer to the description of that command for details.

Address Offset	msb							lsb
operating_mode_specific_information for mode STOP ($C5_{16}$)								
00_{16}	<<no additional information – FF$_{16}$ pad bytes if necessary>>							
:	:							

Table 7-24 operating_mode_specific_information for mode STOP ($C5_{16}$)

As shown, there are no additional values defined for the *operating_mode_specific_information* when the mode is STOP.

Address Offset	msb							lsb
	operating_mode_specific_information for mode RECORD OBJECT (56_{16})							
00_{16}	<<no additional information – FF_{16} pad bytes if necessary>>							
:								

Table 7-25 operating_mode_specific_information for mode RECORD OBJECT (56_{16})

As shown, there are no additional values defined for the *operating_mode_specific_information* when the mode is RECORD OBJECT.

Address Offset	msb							lsb
	operating_mode_specific_information for mode REHEARSAL ($C7_{16}$)							
00_{16}	<<no additional information – FF_{16} pad bytes if necessary>>							
:	:							

Table 7-26 operating_mode_specific_information for mode REHEARSAL ($C7_{16}$)

As shown, there are no additional values defined for the *operating_mode_specific_information* when the mode is REHEARSAL.

Address Offset	msb							lsb
	operating_mode_specific_information for mode SUSPENDED (FF_{16})							
00_{16}	reason							
:	<<FF_{16} pad bytes if necessary>>							

Table 7-27 operating_mode_specific_information for mode SUSPENDED (FF_{16})

The *reason* field indicates why this destination or source plug is suspended. It can take any of the following values:

reason	Meaning
01_{16}	D-IN UNLOCKED – The PLL of the subunit is not locked with the input signal.
02_{16}	CAN'T COPY – The copy prohibit flag of input signal is on.
03_{16}	BANDWIDTH EXCEEDED – The disc subunit does not have the capacity to support any more streams, so this plug is not available.
10_{16}	NO MEDIA – There is no media in the subunit, so the plugs are not available for use.
11_{16}	DISC ERROR – Error caused by disc – inserted upside down, the TOC is unreadable, etc, so the plug is not available for use.
12_{16}	MEDIA PROBLEM – The disc may not be recordable, it might be write protected, or recording capacity is used up.
21_{16}	IMPORTING – The subunit is currently in the process of importing a disc.
22_{16}	EXPORTING – The subunit is currently in the process of exporting a disc.
23_{16}	READING TOC – The disc subunit is busy reading the TOC of the disc, so the plug is not available.
24_{16}	WRITING TOC – The disc subunit is busy writing TOC data to the disc, so the plug is not available.
25_{16}	PLAYING – The disc subunit is busy playing, so the plug is not available.
26_{16}	SEARCH – The disc subunit is busy searching, so the plug is not available.
27_{16}	RECORDING – The disc subunit is busy recording, so the plug is not available.
FF_{16}	UNKNOWN – The plug is not available, for an unspecified reason.
all other values	Reserved for future specification.

Table 7-28 *reason* field

If the subunit has two or more reasons for the plug being suspended, it indicates the lowest code from the table above. For example, if the PLL is not locked (01_{16}) and no disc is in the subunit (10_{16}), it indicates "01_{16}" (D-IN UNLOCKED).

7.2.8 Plug Configuration Info Block (88 07_{16})

The *plug_configuration_info_block* specifies the configuration information for a source or destination plug. The configuration is specified by the type of AV object which will be on that plug (audio track, etc.). The following diagram illustrates this structure:

plug_configuration_info_block	
Address Offset	Contents
00 00$_{16}$	compound_length
00 01$_{16}$	
00 02$_{16}$	info_block_type = 88 07$_{16}$ (plug_configuration_info_block)
00 03$_{16}$	
00 04$_{16}$	primary_fields_length
00 05$_{16}$	
00 06$_{16}$	AV_object_type
00 07$_{16}$	object_and_plug_type_specific_information
:	
:	

Table 7-29 plug_configuration_info_block

The *compound_length* field specifies the number of bytes for the remainder of this information block (including any nested information blocks which may occur after the last non-info block field). Note that currently there are no nested information blocks shown for this structure, but controllers should be prepared for any blocks to be found while parsing the structure.

The *primary_fields_length* specifies the number of bytes for the remaining non-info block fields of this structure (through the *object_and_plug_type_specific_information* field in this case). All nested info blocks shall appear after this.

The *AV_object_type* field specifies what type of disc subunit AV content object this plug is configured for. This field can have one of the values defined in section 8.1 Disc Subunit Object Types on page 41. This value is the same as the *entry_type* field in the AV object descriptor structures.

The *object_and_plug_type_specific_information* field contains the details of the configuration. This configuration information is set using the appropriate CONFIGURE command (for the source or destination plug). The format and contents of this field depend on both the type of subunit plug (source or destination) and the AV content object type it will be carrying. The type of plug status info block that contains this info block will determine the plug type.

The following sections describe the currently defined configuration information for each of the defined AV object types.

7.2.8.1 Audio Object Type-Specific Destination Plug Configuration Information

When a destination plug is configured for an audio object, the *AV_object_type_specific_configuration_info* field has the following format:

Address Offset	msb							lsb
	AV_object_type_specific_configuration_info for Audio Objects							
00_{16}	audio_synchro_rec	increment_position_number	level_sync_on_off	reserved				
01_{16}	audio_recording_sample_rate							
02_{16}	audio_recording_sample_size							
03_{16}	audio_compression_mode							
04_{16}	audio_recording_channel_mode							
05_{16}	audio_recording_volume							
06_{16}								

Table 7-30 AV_object_type_specific_configuration_info for Audio Objects

The *audio_synchro_rec* bit indicates whether the plug is currently configured to begin recording when triggered by the detection of an audio signal. When this bit is set to 1, then the plug is configured for this feature. If the plug is not configured for this feature, or if the subunit does not support this feature, then this bit shall be set to 0.

If the operating mode for the destination plug is REC pause mode and the audio level rises above a certain threshold for a certain duration, then recording begins. During recording, if the signal drops below a certain threshold for a certain duration, then the plug returns to the REC pause mode (waiting for another trigger). If the subunit receives a command such as STOP, or some other condition arises which prevents recording (running out of space, etc.), then the plug mode is set appropriately, and this bit shall be set to 0.

The *increment_position_number* bit indicates whether the plug is currently configured to increment the position number when the recording operation is paused. When this bit is set to 1, then the plug is configured for this feature. If the plug is not configured for this feature, or if the subunit does not support this feature, then this bit shall be set to 0.

If the operating mode for the destination plug is “increment the position number” and the recording operation is paused, then the subunit increments the position number automatically. If the operating mode for the destination plug is NOT “increment the position number” and the recording operation is paused, then the subunit does NOT increment the position number. The following diagram illustrates this operation:

increment_ position_number	Recording	Recording Pause	Recording
1	position n	position n + 1	position n + 1
0	position n	position n	position n

Figure 7-1 increment the position number

The *level_sync_on_off* bit indicates that the “audio level synchronized position number increment” function is on or off. If this bit is set to 1, the position number is incremented when the mute lasts for several seconds.

The *audio_recording_sample_rate*, *audio_recording_sample_size*, *audio_compression_mode* and *audio_recording_channel_mode* fields all have the same interpretation as defined for the *audio_recording_parameters_info_block*, as described on page 134.

The *audio_recording_volume* field specifies how the audio recording volume is configured. The following table illustrates the relative 2-byte values for this field and their corresponding gain:

audio_recording_volume	
Value	Gain
FFFF$_{16}$	+36dB
:	:
0400$_{16}$	+0dB
:	:
0000$_{16}$	- infinity dB

Table 7-31 audio_recording_volume

$$\text{gain(dB)} = 20\log_{10}(\text{value}/400_{16})$$

The gain is applied to the incoming audio signal before recording to the media. When the subunit does not support this function, then value of audio_recording_volume shall be set to 400_{16}.

7.2.8.2 Digital Still Image Object Type-Specific Destination Plug Configuration Information

Currently, there is no AV_object_type_specific_configuration_info for digital still image objects. So, there is only AV object type information in the object_type_specific_destination plug configuration.

7.2.8.3 Textual Object Type-Specific Destination Plug Configuration Information

Currently, there is no AV_object_type_specific_configuration_info for textual objects. So, there is only AV object type information in the object_type_specific_destination plug configuration.

7.2.8.4 Audio Object Type-Specific Source Plug Configuration Information

When a source plug is currently configured for an audio object, the *AV_object_type_specific_configuration_info* field has the following format:

Address Offset	msb							lsb
audio object type-specific source plug configuration information								
00_{16}	audio_mute	internal_mute_off	reserved					
01_{16}	variable_pitch_value							
02_{16}	reserved				variable_speed_value			
03_{16}								

Table 7-32 audio object type-specific source plug configuration information

The *audio_mute* bit indicates whether external audio muting is in effect (= 1) or not (= 0). If the subunit does not support this feature, then this bit shall be set to 0 (the signal is not muted externally). The *audio_mute* feature is the typical mute found on consumer equipment, to cause the output to be silent.

The *internal_mute_off* bit indicates whether internal muting is off (= 1) or on (= 0). If the subunit does not support the ability to turn off internal muting, then this bit shall be set to 0 (internal muting is always on). The internal muting is adapted to audio signal in the case of trick play (fast forward, fast reverse) in general. The *internal_mute_off* feature is often found on professional or higher-end consumer equipment, for use during editing operations.

The *variable_pitch_value* field indicates the ratio of the output pitch to that of the original pitch (usually with no speed change).

The msb of the *variable_pitch_value* field indicates the plus (higher pitch: msb = 0) or minus (lower pitch: msb = 1) sign, and the other seven bits indicate the rate of pitch change (rising or falling). The unit of variable pitch is 10 cent. This field could indicate from –1270 cent to +1270 cent of pitch variation. The semitone is 100 cent and octave is 1200 cent.

The *variable_speed_value* indicates the ratio of the performance speed of output to the original speed (usually with no pitch change).

The msb of the *variable_speed_value* field indicates the plus (faster speed: msb = 0) or minus (slower speed: msb = 1) sign, and the other eleven bits indicate the percentage of speed change (increasing or decreasing). The unit is 0.1%. This field could indicate from –204.8% to +204.8% of speed variation.

7.2.8.5 Digital Still Image Object Type-Specific Source Plug Configuration Information

When a source plug is currently configured for a digital still image object, the *AV_object_type_specific_configuration_info* field has the following format:

Address Offset	msb							lsb
Digital Still Image object type-specific source plug configuration information								
00_{16}	mute	reserved						

Table 7-33 digital still image object type-specific source plug configuration information

The *mute* bit indicates whether external muting of the output signal is in effect (= 1) or not (= 0).

7.2.8.6 Textual Object Type-Specific Source Plug Configuration Information

When a source plug is currently configured for a textual object, the *AV_object_type_specific_configuration_info* field has the following format:

Address Offset	msb							lsb
textual object type-specific source plug configuration information								
00_{16}	mute	reserved						

Table 7-34 textual object type-specific source plug configuration information

The *mute* bit indicates whether external muting of the output signal is in effect (= 1) or not (= 0).

7.2.9 Synchro Plug Group Configuration Info Block (88 $0B_{16}$)

The *synchro_plug_group_configuration_info_block* is specific to synchro plug group. It has the following format:

synchro_plug_group_configuration_info_block			
Address Offset	Contents		
00 00_{16}	compound_length		
00 01_{16}			
00 02_{16}	info_block_type = 88 $0B_{16}$		
00 03_{16}	(synchro_plug_group_configuration_info_block)		
00 04_{16}	primary_fields_length		
00 05_{16}			
00 06_{16}	all_ mute	reserved	
00 07_{16}	reserved		variable_speed_value
00 08_{16}			

Table 7-35 synchro_plug_group_configuration_info_block

The all_mute bit specifies whether all of the streams of the entire synchronized performance represented by this synchro plug group are muted (= 1) or not (= 0).

The *variable_speed_value* field has the same meaning as described in section 7.2.8.4 Audio Object Type-Specific Source Plug Configuration Information.

7.2.10 Playback Order Configuration Info Block (88 08_{16})

The *playback_order_configuration_info_block* contains information about the **current** playback order configuration. The data in this structure reflects one of the following situations:

The true playback order configuration as set by the controller using the CONFIGURE command

A subunit-specified override of the configuration, to accommodate the current operating mode or AV content object(s) being played

As an example, if the controller has set a configuration and then issues the PLAY command for an audio track, then the configuration will be used.

However, if the controller issues the OBJECT NUMBER SELECT command to cause a digital still image to be output, then some of the playback configuration parameters may not be valid. In this case, the subunit must change the playback configuration to accommodate the current operating mode or data type, and report this configuration in the status descriptor.

If the subunit does NOT update the configuration information to reflect its true operating state, then controllers will incorrectly report this state to the user. The result is that the user may see the subunit configured to operate in "repeat mode", but when it gets to the end of the digital still image object, it stops. This can be confusing and frustrating for the user.

After the AV object has been played, the subunit implementation may choose to go back to the previous (controller-specified) configuration, or to remain in the newly established configuration. It is strongly recommended that the previous configuration be re-established.

Address Offset	msb							lsb
00 00$_{16}$	compound_length							
00 01$_{16}$								
00 02$_{16}$	info_block_type = 88 08$_{16}$ (playback_order_configuration_info_block)							
00 03$_{16}$								
00 04$_{16}$	primary_fields_length							
00 05$_{16}$								
00 06$_{16}$	playback_order							
00 07$_{16}$	repeat_mode		reserved					
00 08$_{16}$	track_boundary_operation		reserved					

Table 7-36 playback_order_configuration_info_block

The *compound_length* field specifies the number of bytes for the remainder of this information block (including any nested information blocks which may occur after the last non-info block field). Note that currently there are no nested information blocks shown for this structure, but controllers should be prepared for any blocks to be found while parsing the structure.

The *primary_fields_length* specifies the number of bytes for the remaining non-info block fields of this structure (through the field 00 08$_{16}$ in this case). All nested info blocks shall appear after this.

The *playback_order* field specifies how playback is occurring (or will occur), and is encoded as follows:

playback_order	Meaning
00_{16}	In order – Playback the tracks in order of their occurrence in the specified list.
01_{16}	Shuffle – Play each track once in random order, then stop.
02_{16}	Random – Play each track in random order, continue playing indefinitely.
all other values	Reserved for future specification.

Table 7-37 playback_order

The *repeat_mode* field specifies how the repeat feature is configured:

repeat_mode	Meaning
00b	Play the list specified in the configuration then stop.
01b	Play the track then stop.
10b	Play an entire list then repeat the list continuously.
11b	Play one track and repeat the track continuously.

Table 7-38 repeat_mode

The *track_boundary_operation_mode* field specifies how the operation control feature is configured:

boundary_operation_mode	Meaning
00b	No special operation at the track boundary.
01b	Pause at the beginning of each track.
10b	Insert a few seconds of “blank” at the beginning of each track (subunit chooses how many seconds).
11b	Pause at the point where the sound starts at the beginning of the track.

Table 7-39 boundary_operation_mode

7.2.11 Audio Level Meter Status Info Block (88 09$_{16}$)

The *audio_level_meter_status_info_block* indicates the audio level of its source plug. The *audio_level_meter_status_info_block* has the following format:

Address Offset	msb							lsb
00 00$_{16}$	compound_length							
00 01$_{16}$								
00 02$_{16}$	info_block_type = 88 09$_{16}$ (audio_level_meter_status_info_block)							
00 03$_{16}$								
00 04$_{16}$	primary_fields_length							
00 05$_{16}$								
00 06$_{16}$	measurement		reserved					
00 07$_{16}$	number_of_channels (n)							
00 08$_{16}$	over	channel_audio_level[0] (31 bits)						
00 09$_{16}$								
00 0A$_{16}$								
00 0B$_{16}$								
:	:							
:	over	channel_audio_level[n – 1] (31 bits)						
:								
:								
:								

Table 7-40 audio_level_meter_status_info_block

The *measurement* field indicates how the audio level is measured, according to the following table:

Measurement	Definition	Meaning
00b	absolute amplitude	Absolute amplitude level of audio which is normalized by full scale level. The full scale level is defined by each subunit. If the bit width of this value is less than 31, the lower bits shall be 0 padded.
all others	reserved for future specification	

Table 7-41 measurement field

The duration of sampling is implementation dependent.

The *number_of_channels* field specifies how many audio channels are indicated.

The *channel_audio_level[x]* fields each indicate the audio level of a channel, which is described in 31 bits. In the case of stereo, *channel_audio_level[0]* shall be the level of the left channel and *channel_audio_level[1]* shall be the level of the right channel.

The *over* bit shall set to 1 if the audio level is over full scale.

7.2.12 Monitor Status Info Block (88 0A$_{16}$)

The *monitor_status_info_block* indicates whether the source plug is currently monitoring a destination plug (e.g. listening to what is being recorded). It has the following format:

Address Offset	msb							lsb
00 00$_{16}$	compound_length							
00 01$_{16}$								
00 02$_{16}$	info_block_type = 88 0A$_{16}$ (monitor_status_info_block)							
00 03$_{16}$								
00 04$_{16}$	primary_fields_length							
00 05$_{16}$								
00 06$_{16}$	monitor	reserved						
00 07$_{16}$	destination_plug_number							

Table 7-42 monitor_status_info_block

The *monitor* bit indicates whether the source plug is currently monitoring a destination plug (= 1) or not (= 0).

If the *monitor* bit = 1, then the *destination_plug_number* field indicates the destination plug being monitored. If *monitor* = 0, then *destination_plug_number* shall be set to FF$_{16}$.

8. Disc Subunit Objects

All AV/C disc subunit objects use the general AV/C object format as described in the references at the beginning of this document. The important information for this specification is the *entry_specific_information*, which is described for each disc subunit object. The reader is encouraged to review the other reference material for an overall understanding of how these data structures fit into the AV/C object and object list model.

The general AV/C object descriptor uses one byte for the *entry_type* field, which describes the object. In order to provide additional information which describes the exact nature of the object, each disc subunit object contains additional information that should be used to resolve the specific AV data type.

NOTE: A controller may influence the transmission format of an AV object (audio track, etc.) by establishing a connection from the subunit source plug to a specific unit output plug (either serial bus, analog, etc.). The controller can examine the unit identifier descriptor to determine which transmission format(s) are supported on each unit plug. When the connection is established, the necessary format conversion takes place.

8.1 Disc Subunit Object Types

The following basic *entry_type* values are defined for the disc subunit model:

Disc Subunit Object entry_type Definitions		
Entry Type	Value	Meaning
Audio Track	80_{16}	This object represents an audio track.
Digital Still Image	81_{16}	Objects of this type represent a single still image, such as might be obtained from a digital still camera.
Textual Object	82_{16}	This object represents a large block of textual data, such as the lyrics for an audio track.
Child Directory Object	90_{16}	This object holds the child list ID of a Child Contents List, used to construct the hierarchical file system for subunits which support it.
Performance Object	91_{16}	This object contains a reference to an object in the contents list hierarchy, and associated performance data for that object.
Synchronized Performance Object	92_{16}	Specifies the performance order of several Performance Objects.
Text Database Object	93_{16}	This object represents a small piece of text, such as the title of an audio track.
------	all other values in the subunit-specific range	Reserved for future definition. See the note below regarding the range 80_{16} to $8F_{16}$.

Table 8-1 Disc Subunit Object entry_type Definitions

NOTE: Object types in the range 80_{16} through $8F_{16}$ are often referred to as "AV content objects" in this document. All object entry types which represent AV content stored on the media are AV content objects. Additional AV content object types may be defined in the future.

8.1.1 General Disc Subunit Object entry_specific_information

All disc subunit objects share the following basic format for their *entry_specific_information* field:

Disc Subunit Object entry_specific_information	
Address Offset	Contents
00 00_{16}	non_info_block_fields_length
00 01_{16}	
00 02_{16}	disc_subunit_object_attributes
00 03_{16}	object_type_specific_non_info_block_fields
:	
:	
:	optional info block
:	
:	

Table 8-2 Disc Subunit Object entry_specific_information

The *non_info_block_fields_length* field specifies the number of bytes for the following non-info block fields, which extends through the *object_type_specific_non_info_block_fields* area. If any nested info blocks are present, they will be added after this area.

The *disc_subunit_object_attributes* field specifies a set of attributes that are common to all disc subunit objects, as shown in the following table:

Attribute Bit	Attribute Name	Meaning
1xxx xxxx	has_more_attributes	If this bit is set to 1, then the next byte is also an attributes byte. If this bit is 0, then the next byte is as defined for this structure.
xxxx xxx1	content_locked	1 = the AV content object represented by this descriptor is locked 0 = the AV content object represented by this descriptor is not locked
xxxx xx1x	descriptor_locked	1 = the descriptor is locked 0 = the descriptor is unlocked
all others	------	Reserved for future definition.

Table 8-3 disc_subunit_object_attributes field

The *content_locked* bit indicates whether the AV content object represented by this descriptor is locked. When an object is locked, the object can not be erased or modified (e.g. editing commands such as DIVIDE and COMBINE are not allowed).

The exception to this rule is that locked objects can still be rearranged with the MOVE command.

On non-recordable media, this bit shall be set to 1 (locked) because the object can't be modified.

The *descriptor_locked* bit indicates whether this descriptor structure is locked or not. If locked, then it can't be modified. If unlocked, then it can be modified. It's possible that the descriptor can be modified independently of the AV content object, depending on the subunit implementation.

The *object_type_specific_non_info_block_fields* contains information which is unique to the type of disc subunit object which is represented by this descriptor. The details of these fields for each of the defined object types is in the following sections.

The *nested_info_blocks* area includes zero or more info blocks, depending on the type of object and on the subunit implementation. The controller can determine if any nested info blocks exist based on the following formula:

> if size_of_entry_specific_information > (non_info_block_fields_length + 2) then info blocks exist

> The size_of_entry_specific_information field is in the general object descriptor structure, as specified in reference [1].

The following table lists the info blocks which are common to all disc subunit objects. As with all info block structure definitions, controllers should be prepared to find ANY type of info block in ANY location, and to not treat this as an error (exceptions are noted where applicable). These info blocks are described in section 11 of this document or in the AV/C General Specification.

Info Block Types for the disc subunit object entry_specific_information		
info block type	info block name	meaning for disc subunit objects in general
00 04$_{16}$	time_stamp_info_block (content creation time)	creation time of the object
00 05$_{16}$	time_stamp_info_block (content modification time)	modification time of the object
00 06$_{16}$	time_stamp_info_block (descriptor creation time)	creation time of the object descriptor
00 07$_{16}$	time_stamp_info_block (descriptor modification time)	modification time of the object descriptor
80 04$_{16}$	AV_content_identifier_info_block	a unique identifier for this AV content object, assigned by the creator
00 01$_{16}$	size_indicator_info_block	the size of the content object
00 0B$_{16}$	name_info_block(s)	the title of the object
80 00$_{16}$	artist_info_block(s)	information about the artist(s) who created the object
80 01$_{16}$	genre_info_block	describes the content genre of the object (Jazz, Classical, Mixed, etc.)
00 0D$_{16}$	image_info_block(s)	image(s) representing the object

Table 8-4 Info Block Types for the disc subunit object entry_specific_information

8.2 Audio Object entry_specific_information

An audio object represents an audio track on the disc media. It has the following *entry_specific_information*:

Audio Track Object entry_specific_information	
Address Offset	Contents
00 00$_{16}$	non_info_block_fields_length
00 01$_{16}$	
00 02$_{16}$	disc_subunit_object_attributes
00 03$_{16}$	audio_recording_parameters_info_block
:	
:	
:	optional info blocks
:	
:	

Table 8-5 Audio Track Object entry_specific_information

The *non_info_block_fields_length* and *disc_subunit_object_attributes* fields are as described above for the general disc subunit *entry_specific_information* fields.

The *audio_recording_parameters_info_block* specifies the parameters used for recording this audio object. For details, refer to that info block description on page 134.

The inclusion of additional info blocks, as noted by the nested_info_blocks field, is optional. The *size_indicator_info_block*, *name_info_block*, *genre_info_block* and *artist_info_block* are recommended for inclusion, if the media and subunit implementation support them. Other possible info blocks are any of those shown in the table at the beginning of this section; however, controllers should always be prepared to find any type of info blocks here, and to not treat this as an error.

8.3 Digital Still Image Object entry_specific_information

The digital still image object descriptor specifies an image. This image can be a picture, icon, etc. The entry_specific_information is defined as follows:

Digital Still Image entry_specific_information	
Address Offset	Contents
00 00$_{16}$	non_info_block_fields_length
00 01$_{16}$	
00 02$_{16}$	disc_subunit_object_attributes
00 03$_{16}$	image_format_info_block
:	
:	
:	size_indicator_info_block (raw_byte_count format)
:	
:	
:	optional info blocks
:	
:	

Table 8-6 Digital Still Image entry_specific_information

The *non_info_block_fields_length* and *disc_subunit_object_attributes* fields are as described above for the general disc subunit *entry_specific_information* fields.

The *image_format_info_block* field describes the format of the image. This info block is defined in reference [1].

The *size_indicator_info_block* specifies the size, in bytes, of this digital still image.

Additional info blocks might be found in this structure, as noted by the “other optional info blocks” field. Controllers should be ready to discover any info blocks here and to not treat this as an error condition.

8.4 Textual Object entry_specific_information

A textual object descriptor provides descriptive information about a text-based object on the disc media. Examples of textual objects include the lyrics for an audio track, a general text file, etc. The object descriptor does NOT contain the text itself. The *entry_specific_information* is as follows:

Textual Object entry_specific_information	
Address Offset	Contents
$00\ 00_{16}$	non_info_block_fields_length
$00\ 01_{16}$	
$00\ 02_{16}$	disc_subunit_object_attributes
$00\ 03_{16}$	size_indicator_info_block
:	
:	
:	character_code_info_block
:	
:	
:	language_code_info_block
:	
:	
:	file_format_info_block
:	
:	
:	text_content_type_info_block
:	
:	
:	optional info blocks
:	
:	

Table 8-7 Textual Object entry_specific_information

The *size_indicator_block* specifies the size, in bytes, of the textual object. This info block is mandatory.

The *character_code_info_block* and *language_code_info_block* structures specify the character and language codes for the textual object. These two info blocks are optional; if they appear, they must appear in the order shown in the diagram. If they are left out, then the textual object is encoded in minimal ASCII English.

The *file_format_info_block* specifies the format of the file which holds this textual object. For details, refer to that info block description.

The *text_content_info_block* provides an encoded value that indicates what the content of this textual object represents (e.g., lyrics, liner notes, etc.). Note that it would also be possible to (optionally) include a *description_info_block* that provides a human-readable description of the contents of this textual object (e.g. “lyrics”). The *text_content_info_block* is mandatory.

Other optional info blocks that might be included are shown in the table above, in section 8.1.1. Controllers should expect to find any type of info block, and not treat this as an error.

Additionally, the above info blocks shown as mandatory do not necessarily have to appear in any particular order, except for the character and language code info blocks as mentioned above. Controllers should not assume the ordering of the other mandatory info blocks.

Note that unlike text database objects and text info blocks, there are no attributes such as “user-modifiable” and “stored on media” for textual objects, because these are content objects. They are treated the same way as audio tracks, video tracks, etc.

8.5 Child Directory Object entry_specific_information

The child directory object exists only to hold the ID of a child list, which represents a subdirectory. The ID is placed in the *child_list_ID* field of the object descriptor structure (not in the *entry_specific_information* field).

Child Directory Object entry_specific_information	
Address Offset	Contents
00 00$_{16}$	non_info_block_fields_length
00 01$_{16}$	
00 02$_{16}$	disc_subunit_object_attributes
:	optional info blocks
:	
:	

Table 8-8 Child Directory Object entry_specific_information

Restrictions on info blocks for the child directory object: the only info blocks which currently make sense to include in the child directory *entry_specific_information* would be the *time_stamp_info_block* structures which indicate the descriptor creation and modification dates. All other info blocks that are used to describe the media contents should be in the child list header.

Controllers should be designed to allow **any** type of info block to be found in this area, to allow for future expansion.

8.6 Performance Object entry_specific_information

Performance objects contain a reference to an AV content object (audio track, digital still image, etc.), and associated performance data. For details on how these objects are used, please refer to section 9.6 Performance Lists on page 65. The *entry_specific_information* is defined as follows:

Performance Object entry_specific_information	
Address Offset	Contents
00 00$_{16}$	non_info_block_fields_length
00 01$_{16}$	
00 02$_{16}$	disc_subunit_object_attributes
00 03$_{16}$	descriptor_reference_info_block (AV content object reference)
:	
:	
:	output_start_time -OR- presentation_start_time (position_indicator_info_block)
:	
:	
:	presentation_end_time (position_indicator_info_block)
:	
:	
:	content_entry_point (position_indicator_info_block)
:	
:	
:	content_exit_point (position_indicator_info_block)
:	
:	
:	optional info blocks
:	
:	

Table 8-9 Performance Object entry_specific_information

The *descriptor_reference_info_block* points to an AV content object in the contents (or temporary contents) hierarchy. As described in the section that details the disc subunit object structures, some objects describe content on the media (audio tracks, etc.).

See the definition of this info block for more details. This info block is required.

Generally, the format of this descriptor reference may be any of the defined object reference types, but there may be media-format restrictions. For details on media-specific restrictions, refer to the appropriate disc subunit media format specification.

The info block shown as *output_start_time* –OR– *presentation_start_time* can be ONE of these two attributes (but not both). Each of them are *position_indicator_info_block* structures. These two have very different meanings in the case of a file-type of transfer, such as a digital still image.

The *output_start_time* block can be in the format *absolute_HMSF_count* or *absolute_clock_time*. It specifies the time, based on the beginning of the output transmission of the performance, to begin transmitting the content object referred to by the *descriptor_reference_info_block* on the source plug. This info block is optional; if it is not present, then the content object shall be transmitted at the beginning of the performance.

The following diagram illustrates how this info block is to be treated in the case of a file-type object such as a digital still image (DSI):

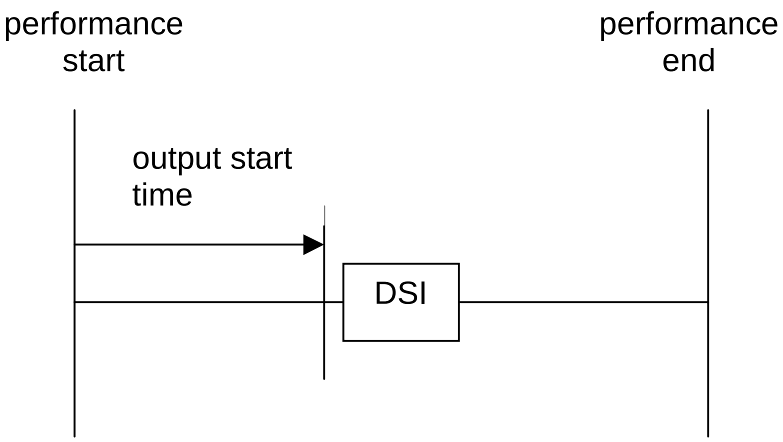


Figure 8-1 output start time

As shown, the bytes of the DSI object begin transmitting immediately on (or as soon as possible after) the output start time.

The *presentation_start_time* block can be in the format *absolute_HMSF_count* or *absolute_clock_time*. It specifies the *intended* time for desplay device to present the content object to the user. The disc subunit should begin transmitting the object before this time, to allow for transmission and processing of the data at the display/presentation side. The choice of when to begin transmitting is decided by the subunit implementation. Of course there can be no guarantee that the object will be completely displayed at the target time; the presentation method is an implementation matter for the destination device. This info block is also optional.

The disc subunit which has no knowledge of system delay may begin transmitting on the presentaition start time.

The following diagram illustrates how this info block is to be treated in the case of a file-type object such as a digital still image (DSI):

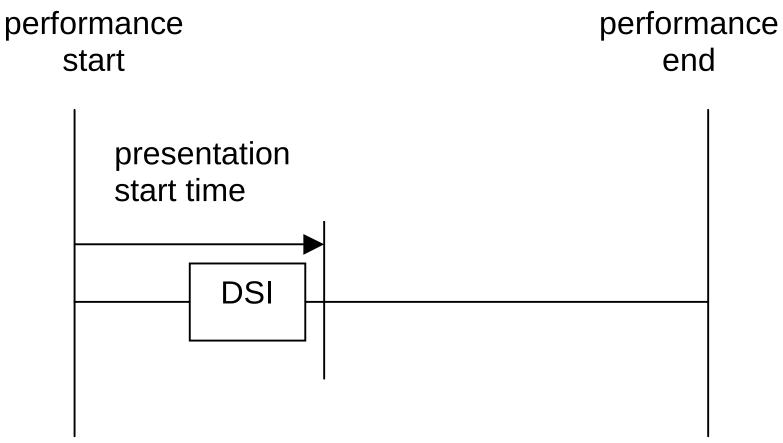


Figure 8-2 presentation start time

As shown, the bytes of the DSI object begin transmitting some time before the intended presentation start time.

The *presentation_end_time* block can be in the format *absolute_HMSF_count* or *absolute_clock_time*. It specifies the *recommended* time to stop presenting the content object to the user. Whether the presentation device actually stops presenting the object at this time, or not is an implementation choice of the presentation device. This info block is optional.

The following diagram illustrates how this info block is to be treated in the case of a file-type object such as a digital still image (DSI):

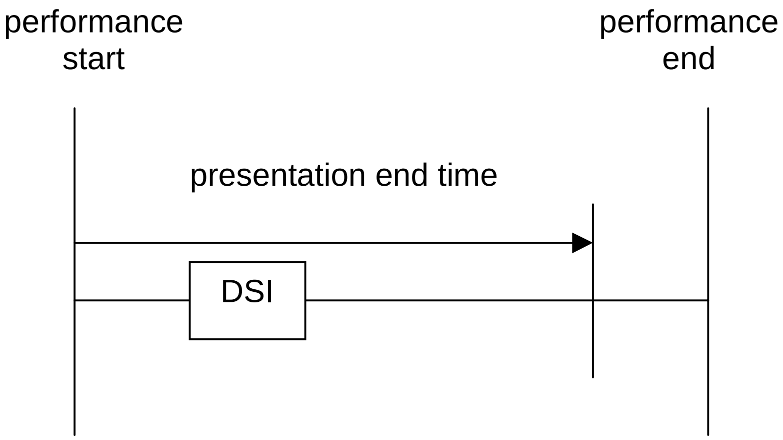


Figure 8-3 presentation end time

A *presentation_end_time* of all FF_{16} bytes has a special meaning: unspecified. In the main performance list, this means that the *presentation_end_time* is the same as the performance end time. In a child performance list, it means that the *presentation_end_time* is the same as the *presentation_start_time* of the next performance object. This is illustrated in the diagram below:

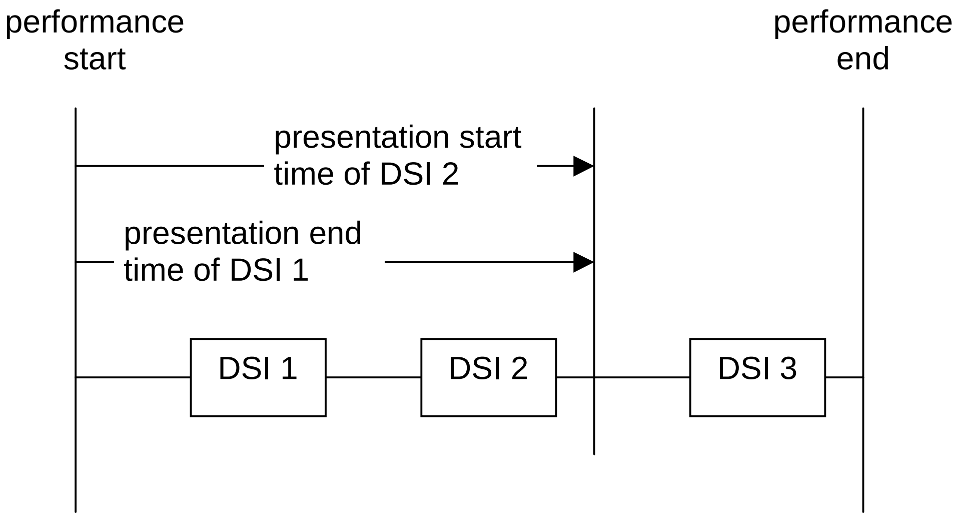

performance start
performance end
presentation start time of DSI 2
presentation end time of DSI 1
DSI 1
DSI 2
DSI 3

Figure 8-4 presentation end time

If the performance object contains an *output_start_time* info block for DSI 1, then a *presentation_end_time* of all FF_{16} bytes means that the presentation end time is the same as the *presentation_start_time* of DSI 2.

The *content_entry_point* and *content_exit_point* specify “trim” information – where, in the content object, to start and stop playing. These values are measured from the beginning of the content object. They are specified as *position_indicator_info_block* structures, as formats 02_{16} (hours:minutes:seconds:frames), 04_{16} (byte count), or 07_{16} (clock time). The specification format will depend on the type of content object and the subunit implementation’s support for the format. To specify a *content_entry_point* at the beginning of the item, use values of 00_{16} for all of the appropriate entries. To specify a *content_exit_point* at the end, use values of FF_{16}. The behavior is undefined if the entry point is greater than the exit point.

These blocks are also optional. If they are left out, then the entire object shall be played.

Restrictions on info blocks: the info blocks other than descriptor_reference_info_block are optional. Controllers should be designed to allow the discovery of other info block types without treating this as an error.

Section 9.6 explains the concepts behind performance lists. A very brief explanation is provided here to give some context for how the output and presentation start/end times are dealt with.

There are two kinds of performance lists: main performance lists and child performance lists.

The main performance list contains child directory objects and/or performance objects. Each entry (whether it is a child directory or performance object) represents ONE performance. The performances are played individually. The following diagram illustrates this concept:

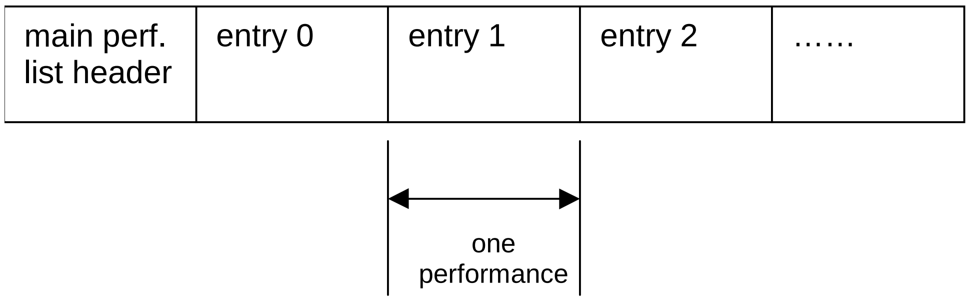


Figure 8-5 one performance

If a performance consists of two or more performance objects, then that performance will be described with a child performance list. The entries in a child performance list are all performance objects (there are no further levels of child lists in the hierarchy). The following diagram illustrates this concept:

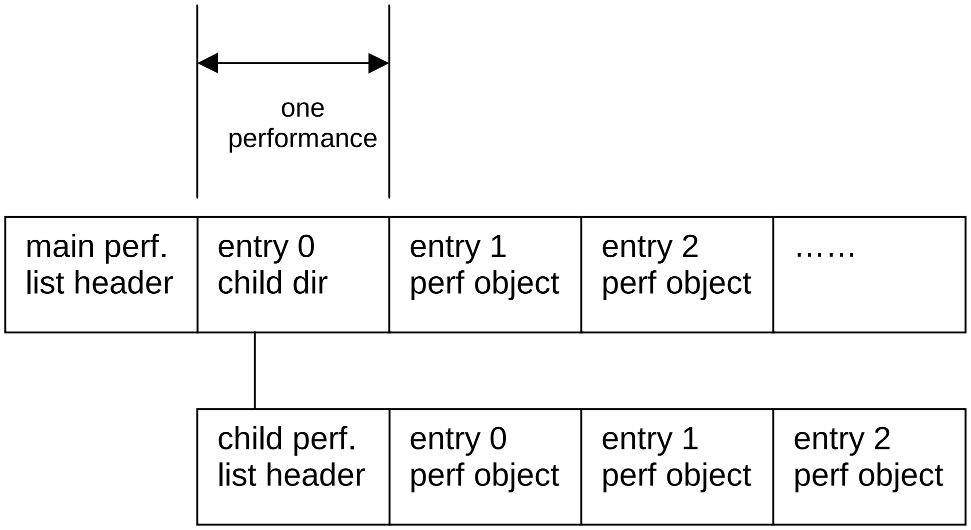


Figure 8-6 one performance

In the child performance list, the *output_start_time*, *presentation_start_time* and *presentation_end_time* are measured from the beginning of the performance (e.g. from the beginning of the child list). Each object in the list is played sequentially for the performance.

8.7 Synchronized Performance Object entry_specific_information

Synchronized performance objects contain a “performance specifier”, which indicates the position of a performance object in a performance list. Several synchronized performance objects can be used to specify the order in which performance objects are played. For more details, please refer to section 9.6 Performance Lists on page 65. The *entry_specific_information* for synchronized performance objects is as follows:

Synchronized Performance Object entry_specific_information	
Address Offset	Contents
00 00_{16}	non_info_block_fields_length
00 01_{16}	
00 02_{16}	disc_subunit_object_attributes
00 03_{16}	performance_specifier
:	
:	
:	optional info blocks
:	
:	

Table 8-10 Synchronized Performance Object entry_specific_information

The *performacne_specifiere* contains position number that specifies the position of a performance object in a list. The number of bytes used for the perfromance_specifier is specified by the *size_of_object_position* field in the disc subunit identifier descriptor. There may be any number of other info blocks included, such as an image or name, description, etc. Controllers should be prepared to find any number and type of info blocks, and to not treat this as an error condition.

8.8 Text Database Object entry_specific_information

Text database objects **contain** small pieces of text, such as the names of objects, which can be referred to by other descriptors. This helps to keep the size of other descriptors more stable, because they don't embed text strings which may often change in length. For more details, please refer to section 9.8 Text Database Lists on page 80. The format of the *entry_specific_information* is as follows:

Text Database Object entry_specific_information	
Address Offset	Contents
00 00_{16}	non_info_block_fields_length
00 01_{16}	
00 02_{16}	disc_subunit_object_attributes
00 03_{16}	text_database_content_attributes_info_block
:	
:	
:	character_code_info_block
:	
:	
:	language_code_info_block
:	
:	
:	raw_text_info_block
:	
:	
:	other optional info blocks
:	
:	

Table 8-11 Text Database Object entry_specific_information

The *text_database_content_attributes_info_block* describes the characteristics of the text database entry referred to by this object descriptor. For details, refer to that info block description. This info block is required.

The character and language code info blocks describe the format of the text in the *raw_text_info_block*. These two info blocks are optional; if they are specified, then they must be in the order shown. If they are not included, then the text is assumed to be in minimal ASCII English format.

The *raw_text_info_block* contains the text; this info block is mandatory.

The other optional info blocks can be chosen from the table of disc subunit object info blocks shown at the beginning of this section. Note that the size_indicator block doesn’t make sense, because this object does not refer to an AV content object. However, controllers should generally be prepared to find ANY type and number of info blocks in any location at any time, and to not treat this as an error.

9. Disc Subunit Object Lists

REMINDER: In order to fully understand the information presented in this section, it is necessary to understand the general AV/C object and object list concepts which are described in reference [1].

9.1 Disc Subunit List Types

The following *list_type* values are defined for the AV/C disc subunit. The table provides a brief description of the lists, but further details are provided in subsequent sections of this document:

list name	list_type	comments
Root Contents List	80_{16}	This list contains information about the installed disc, and it contains objects that are at the “root” storage level of the disc. This is a root list, whose ID is in the subunit identifier descriptor.
Child Contents Lists	81_{16}	These lists exist as child lists (subdirectories) in the hierarchy topped by the root contents list, and only exist if the subunit supports a hierarchical storage model. These lists do not contain disc information.
Root Temporary Contents List	82_{16}	This list contains information about the installed disc, and it contains objects that are at the “root” storage level of the disc. It is used for editing purposes. This is a root list, whose ID is in the subunit identifier descriptor.
Child Temporary Contents Lists	83_{16}	These lists exist as child lists (subdirectories) in the hierarchy topped by the root contents list, and only exist if the subunit supports a hierarchical storage model. These are also used for editing purposes. These lists do not contain disc information.
Performance Lists	84_{16}	These lists contain performance objects, which are references to AV objects in the contents lists, specifying a set of performance characteristics for each item. A performance object can specify the playback of several content items. The root, main and child lists have the same structure.
Synchronized Performance Lists	85_{16}	This list contains references to performance objects, specifying a certain playback order for collections of performances. The root and child lists have the same structure.
Text Database Lists	86_{16}	This list contains text database objects.
------------	all others in the subunit-specific range	Reserved for future specification.

Table 9-1 disc subunit list type

9.2 General Disc Subunit List list_specific_information

All disc subunit lists share the following basic format for their *list_specific_information* field:

Disc Subunit List list_specific_information	
Address Offset	Contents
00 00_{16}	non_info_block_fields_length
00 01_{16}	
00 02_{16}	disc_subunit_list_attributes
00 03_{16}	list_type_specific_non_info_block_fields
:	
:	
:	nested_info_blocks (optional or required)
:	
:	

Table 9-2 Disc Subunit List list_specific_information

The *non_info_block_fields_length* field specifies the number of bytes for the following fields, which extends through the *list_type_specific_non_info_block_fields* area. If any nested info blocks are present, they will be added after this area.

The *disc_subunit_list_attributes* field specifies a set of attributes that are common to all disc subunit lists, as shown in the following table:

Attribute Bit	Attribute Name	Meaning
1xxx xxxx	has_more_attributes	If this bit is set to 1, then the next byte is also an attributes byte. If this bit is 0, then the next byte is as defined for this structure.
xxxx xxx1	content_locked	1 = the AV content objects represented by the object descriptors in this list are locked 0 = the AV content objects represented by the object descriptors in this list are not locked
xxxx xx1x	descriptor_locked	1 = the list descriptor and all object descriptors inside are locked 0 = the list descriptor is unlocked –some object descriptors may be locked or unlocked; the controller must examine them individually
all others	------	Reserved for future definition.

Table 9-3 disc_subunit_list_attributes field

The *content_locked* bit indicates whether the AV content objects represented by the object descriptors in this list are locked. When objects are locked, they can not be erased or modified (e.g. editing commands such as DIVIDE and COMBINE are not allowed). This attribute bit is a convenient way for controllers to quickly determine if any of the content objects represented by this list can be modified. If the list does not contain content object references, then this bit shall be set to 1 (indicating that there is no modifiable content referenced by the list).

The exception to this rule is that locked objects can still be rearranged with the MOVE command.

On non-recordable media, this bit shall be set to 1 (locked) because the objects can’t be modified.

The *descriptor_locked* bit indicates whether this list structure and all of the object descriptor structures it contains are locked or not. If locked, then neither the list nor any of the object descriptors it can be modified. If unlocked, then the list descriptor can be modified. It's possible that some of the object descriptor structures it contains may be locked; controllers will have to examine them individually to determine this. This bit is convenient for controllers to quickly determine if they are not able to modify any of the descriptor contents in the list structure.

The *list_type_specific_non_info_block_fields* contains information which is unique to the type of disc subunit list which is represented by this descriptor. The details of these fields for each of the defined list types is defined below.

The *nested_info_blocks* area includes zero or more info blocks, depending on the type of list and on the subunit implementation. The controller can determine if any nested info blocks exist based on the following formula:

> if size_of_list_specific_information > (non_info_block_fields_length + 2) then info blocks exist.

Each list type description below indicates the info blocks which are either required or optional. As with all info block structure definitions, controllers should be prepared to find ANY type of info block in ANY location, and to not treat this as an error (exceptions are noted where applicable).

9.3 Root Contents List

The root contents list represents a disc (CD-DA, MD etc.) that is currently installed in the disc subunit. The list contains information describing the contents of the disc as a whole (such as the disc title and image), as well as a collection of objects which represent AV content (audio tracks, video tracks, digital still images, etc.). If the subunit supports only a flat content storage model, then the objects in this list represents the entire AV contents area of the media. If the subunit supports a hierarchical data storage model, then there may be any number of additional lists, which are of the Child Contents List type. These lists are explained in more detail below.

The following diagram illustrates how a flat storage system would be represented (using only the Root Contents List):

Root Contents List Header	Audio Track	Video Track	Audio Track	Digital Still Image Track	Digital Still Image Track

Figure 9-1 flat storage system

In the above example, the various AV objects on the disc are represented by the object entries in the root contents list. This represents the entire AV content of this particular disc. For the conceptual diagram above, the root contents list header is referring to the common header shared by all AV/C list types, in addition to the *list_specific_information* area described below. In other words, the list header is everything in the list except the objects.

9.3.1 Root Contents List list_specific_information

The root contents list *list_specific_information* contains “global” information about the disc. The *list_specific_information* field for the root contents list has the following format:

Root Contents List list_specific_information	
Address Offset	Contents
00 00$_{16}$	non_info_block_fields_length
00 01$_{16}$	
00 02$_{16}$	disc_subunit_list_attributes
00 03$_{16}$	media_type
00 04$_{16}$	
00 05$_{16}$	disc_recordable_information
00 06$_{16}$	time_stamp_info_block (descriptor modification time)
:	
:	
:	default_play_list_info_block
:	
:	
:	other optional info blocks
:	
:	

Table 9-4 Root Contents List list_specific_information

The *non_info_block_fields_length* specifies the size, in bytes, of the non-info block fields (through the *disc_recordable_information* field in this case).

The *disc_subunit_list_attributes* field is as defined above.

The *media_type* field indicates the format of the information on this disc. The upper byte indicates the media family, while the lower byte specifies more detailed information. It is encoded as follows:

media_type (MSB)	Value	media_type (LSB)	Value
CD	01$_{16}$	CD-DA reserved for Video-CD other	01$_{16}$ 02$_{16}$ 0E$_{16}$
MD	03$_{16}$	MD-audio reserved for MD-picture other	01$_{16}$ 02$_{16}$ 0E$_{16}$
unknown	FF$_{16}$	unknown	FF$_{16}$
all others	reserved	reserved	reserved

Table 9-5 media_type field

The *other* value for *media_type* (the LSB) means that the disc is something other than a recognized AV format (such as a CD-ROM). When this value is reported, it indicates that the subunit is able to recognize the disc but it contains information which is not recognizable to the subunit.

The *disc_recordable_information* field format is as follows:

address offset	msb						lsb
	disc_recordable_information						
00_{16}	protected		recordable		reserved		

Table 9-6 disc_recordable_information field

The *protected* field indicates whether the disc is protected from recording or not. If this field is set to 01_{16}. then the disc is protected. If this bit is set to 00_{16}, then the disc is not protected. If this bit is set to 10_{16} then the write-protect information is unknown. The value 11_{16} is reserved for future specification.

As an example of its use, this field shows the state of the write-protect tab on a MiniDisc.

The *recordable* field indicates whether the disc is recordable or not. If this field is set to 01_{16}, the disc is recordable. If this field is set to00_{16}, the disc is not recordable. If this field is set to 10_{16} then the write-protect information is unknown. The value 11_{16} is reserved for future specification.

The *protected* and *recordable* fields are mutually exclusive; even if a disc is write-protected, the recordable field shall be set to indicate the possibility of recording, based on the nature of the media type (e.g. an MD-audio disc has the possibility of being recorded, if the write protection is removed).

The *time_stamp_info_block* indicates the time stamp when this list was last modified. Its format is specified in reference[1]. This info block is required.

> **NOTE:** The *time_stamp_info_block* that describes when the list was modified should not be updated if the only change is due to a “continuously changing” value such as a position counter or capacity value. This minimizes wasted time and resources for controllers who want to check the modification time to quickly determine if a “meaningful” change has occurred, such as a change in the table of contents for a disc.

The *default_play_list_info_block* specifies which list shall be used as the default for play operations. Refer to the info block description for more details. This info block is required.

The optional info block area contains zero or more additional information blocks. Generally, none of these blocks are required; however, certain media type specifications may place other restrictions or requirements on this area. The following table illustrates the info blocks which might appear in this area, as currently defined. Controllers should not treat the discovery of additional info block types here, depending on the media type or future specification updates.

Info Block Types for the Root Contents List optional_info_block_area		
info block type	info block name	meaning for Root Contents List
80 02$_{16}$	disc_capacity_info_block	the capacity of the disc
80 03$_{16}$	AV_object_type_specific_capacity_info_block	the disc storage area allocated for each AV object type
00 04$_{16}$	time_stamp_info_block (content creation)	disc creation time
00 05$_{16}$	time_stamp_info_block (content modification)	disc modification time
00 06$_{16}$	time_stamp_info_block (descriptor creation)	list creation time
80 05$_{16}$	disc_catalog_code_info_block	disc catalog code
00 0B$_{16}$	name_info_block	the title of the disc
80 00$_{16}$	artist_info_block	information about the artists represented on the disc
80 01$_{16}$	genre_info_block	describes the content genre of the disc (Jazz, Classical, Mixed, etc.)
00 0D$_{16}$	image_info_block	an image representing the disc

Table 9-7 Info Block Types for the Root Contents List optional_info_block_area

9.4 Child Contents Lists

If the disc subunit supports a hierarchical storage system, then all of the child lists below the root contents list shall be of the type *Child Contents List*. A child contents list is used to hold either AV object descriptors (audio track objects, video segment objects, digital still image objects, etc.), or to hold child directory objects. Child directory objects, which are described in section 8.5 on page 47, are used for one purpose only: to hold the list ID of a child list.

The main difference between a root contents list and a child contents list is that the child contents list does NOT have the header information that describes the disc media (such as the disc type, etc.). The following diagram illustrates how a hierarchical storage model could be implemented, using the combination of a root contents list and any number of child contents lists:

Example: A hierarchical storage system
(uses the Root Contents List and Child Contents Lists)

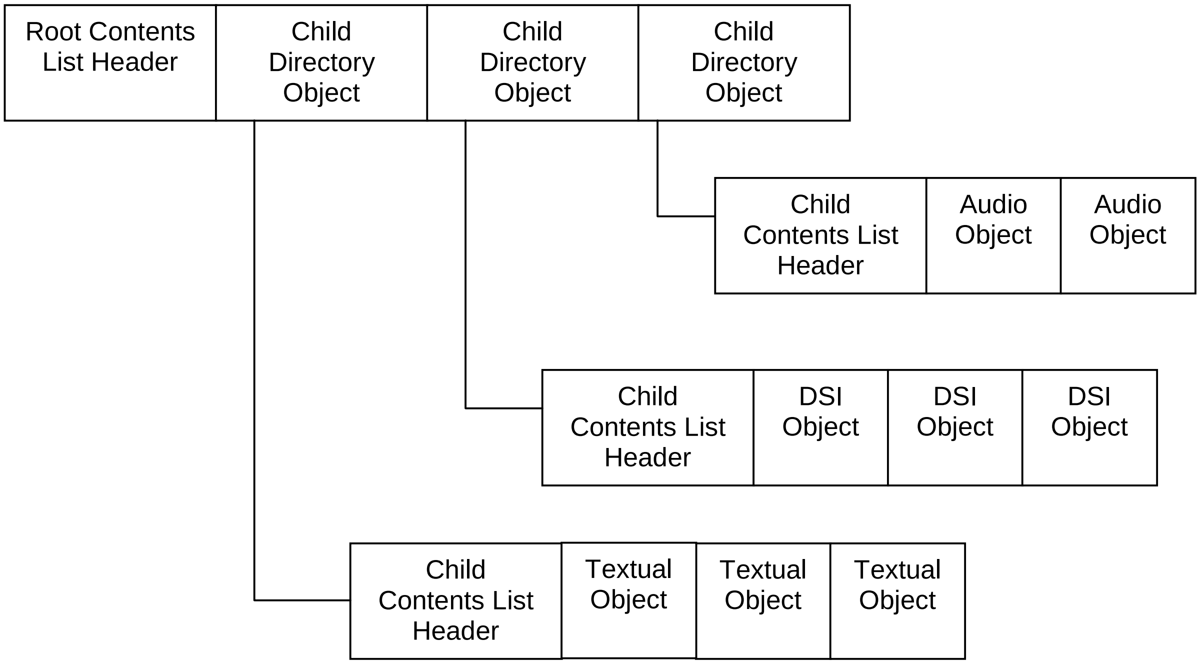


Figure 9-2 hierarchical storage system

9.4.1 Child Contents List list_specific_information

The *list_specific_information* field of a child contents list is as follows:

Child Contents List list_specific_information	
Address Offset	Contents
00 00$_{16}$	non_info_block_fields_length
00 01$_{16}$	
00 02$_{16}$	disc_subunit_list_attributes
00 03$_{16}$	time_stamp_info_block (descriptor modification time)
:	
:	
:	other optional info blocks
:	
:	

Table 9-8 Child Contents List list_specific_information

The *disc_subunit_list_attributes* field is as described above.

The *time_stamp_modification_time* info block indicates when this list was list modified. This is required.

The following table illustrates some of the common info blocks which might be found in the optional info blocks area. Controllers should be prepared to find ANY number of ANY type of info blocks in ANY location (exceptions noted where appropriate), and to not treat this as an error.

Info Block Types for the Child Contents List common_info_block_area		
info block type	info block name	meaning for Child Contents List
00 04$_{16}$	time_stamp_info_block (content creation)	creation time of AV objects represented in the list
00 05$_{16}$	time_stamp_info_block (content modification)	modification time of AV objects represented in the list
00 06$_{16}$	time_stamp_info_block (descriptor creation)	list creation time
00 0B$_{16}$	name_info_block	the name of the list (e.g. name of subdirectory)
80 00$_{16}$	artist_info_block	information about the artists who created the AV content objects described by the list
80 01$_{16}$	genre_info_block	describes the content genre of the list (Jazz, Classical, Mixed, etc.)
00 0D$_{16}$	image_info_block	an image representing the list – such as a subdirectory icon, etc.

Table 9-9 Info Block Types for the Child Contents List common_info_block_area

9.5 Temporary Contents Lists and Objects

9.5.1 Overview of Editing Procedures

If the disc subunit and media type support editing the contents of the media (e.g. DIVIDE, COMBINE), there are two broad methods of maintaining the data as editing changes made:

00 a) Perform all of the changes on the actual data automatically

b) Work with “temporary” data, performing the changes on a copy of the actual data

To support option “b”, **Temporary Contents Lists** are defined. Temporary contents lists have the same general structure as regular contents lists, but are kept in a separate hierarchy.

Note: the structure of the Contents list and the Temporary Contents list is the same; only the *list_type* is different.

The following disc subunit commands are used on the temporary contents lists:

- AUTO UPDATE ON/OFF
- ACCEPT EDITING CHANGES
- REJECT EDITING CHANGES

The following rules are defined:

if only contents lists are supported, then the effects of editing is immediate, and causes changes to the contents lists

if temporary contents lists are supported, then editing causes changes in these lists

editing changes from the temporary to the (regular) contents lists are updated in the following ways:

if AUTO UPDATE is ON, then changes are made automatically to the contents lists

If AUTO UPDATE is OFF, then changes are not automatically propagated to the contents lists; in this case:

if the changes are accepted by the user, the controller should use the ACCEPT/REJECT EDITING CHANGES (accepting the changes) command to cause the changes to be committed to the (regular) contents lists

otherwise, if the changes are rejected by the user, the controller should use the ACCEPT/REJECT EDITING CHANGES (rejecting the changes) command to read the original contents information from the disc back into memory, thus wiping out the temporary list changes

the status of the auto update feature (currently on or off) is in the disc subunit status descriptor

There are 3 possible disc subunit configurations, related to the contents hierarchies:

- contents lists only
- temporary contents lists only
- both temporary contents lists and contents lists

Supporting one of these combinations allows the subunit to manage, and therefore the controller to represent to the user, the contents description data in different ways.

If only the contents lists are supported, then the following diagram illustrates a conceptual view of the model:

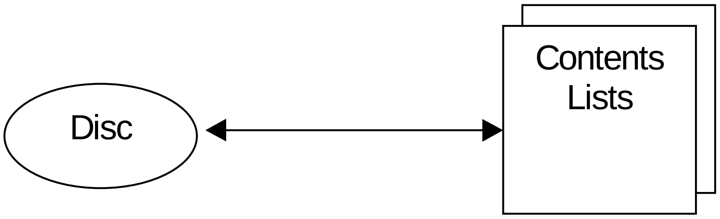


Figure 9-3 contents list

In the contents lists-only situation, the following rules apply:

Editing takes place in the "working area"; all editing changes will automatically affect the disc and will also be reflected in the contents lists, as they occur

There is no way to "buffer" the changes and prevent them from being carried out

A READ DESCRIPTOR command of the contents lists always indicates the actual state of the disc media

If only the temporary contents lists are supported, then the following diagram illustrates a conceptual view of the model:

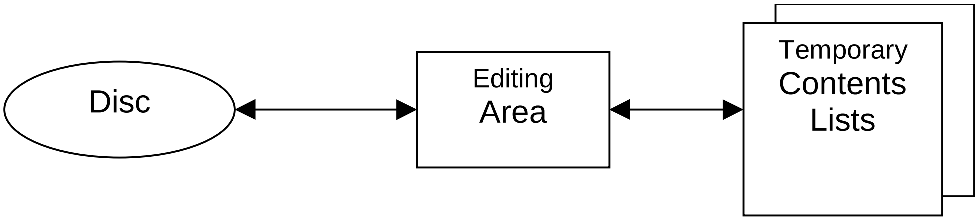


Figure 9-4 temporary contents list

In the temporary contents lists-only situation, the following rules apply:

Editing takes place in the "editing area"; it is possible to buffer the editing changes, to prevent them from affecting the disc and the contents lists, until some time in the future

A READ DESCRIPTOR command of the temporary contents lists indicates the current editing situation of the contents; if the changes have not been written to disc yet, then there is no way for the controller to determine the ACTUAL state of the disc media

If both the temporary and (regular) contents lists are supported, then the following diagram illustrates a conceptual view of the model:

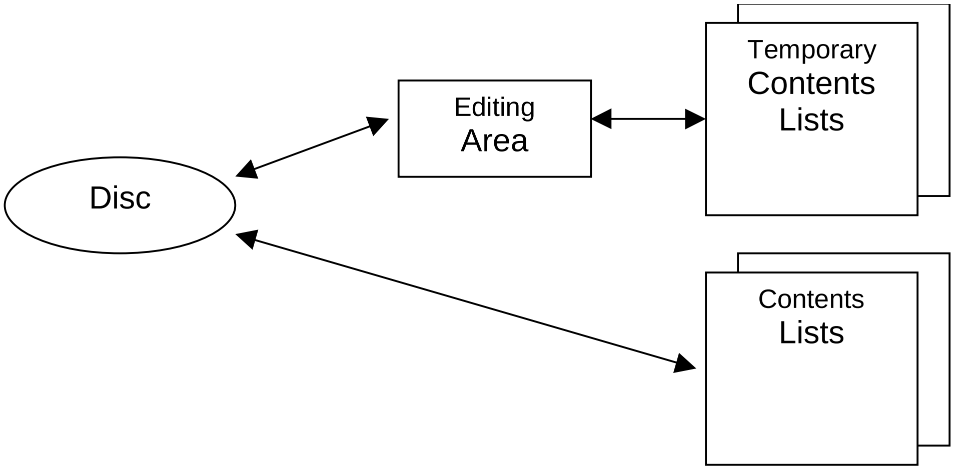


Figure 9-5 contents list and temporary contents list

In this situation, the rules described above still apply for the contents and temporary contents hierarchies. However, while editing actions are being carried out, the controller is still able to access, and present to the user, the true state of the disc media.

9.5.2 Temporary Contents List Hierarchy

The hierarchy of the temporary list structures has the same flexibility as the contents list hierarchy; a single root list and several child lists.

The temporary contents list and (regular) contents list hierarchies are not mixed – all contents lists are in one hierarchy, and all temporary contents lists are in a separate hierarchy.

9.5.3 Temporary Contents Object Structure

Objects in the temporary contents lists are standard AV contents objects (see section 8.1).

9.5.4 Temporary Contents List Structure

The format of the root and child temporary contents lists is the same as for the (regular) contents lists, except for the *list_type* value, as shown in the table above.

9.6 Performance Lists

9.6.1 Overview of Performances

The AV/C disc model allows a controller to play any list of AV objects. Thus, the root contents list or any of the child contents lists can be used for this purpose. However, when contents lists are played, each object is simply played sequentially with no modification; this is the typical user experience of pressing the “Play” button on a CD player.

Some disc subunits may have the ability to tailor the playback, by adjusting the start time of an object on playback, or by playing only a portion of an object (such as the first 10 seconds of an audio track). All of the objects are played in order, but with the modifications described here. The playback of several items sequentially on a single subunit source plug is called a **Performance**. Any of the disc subunit content objects can be used in a performance (audio, textual, digital still image).

To facilitate this concept, a Performance List is defined. The performance list is used to schedule the playback of content on a source plug; it contains one or more objects, which can be of the following types:

a Performance Object

a Child Directory Object

Each performance object represents an individual performance of zero or one content item.

A Child Directory Object, as described in section 8.5, is used to point to a child list which can contain zero or more objects of a given type. In the context of performances, the child directory object is used to point to another performance list, which may contain zero or more performance objects. For the purposes of a performance, the objects under a child directory object are considered to be one “performance”. This is illustrated below in Figure 9-6.

Using the above definition, the performance list can represent several different performances of the content on the disc media.

The items to be performed on a given source plug are scheduled; in this case, scheduling involves the specification, for each item to be performed, of the following information:

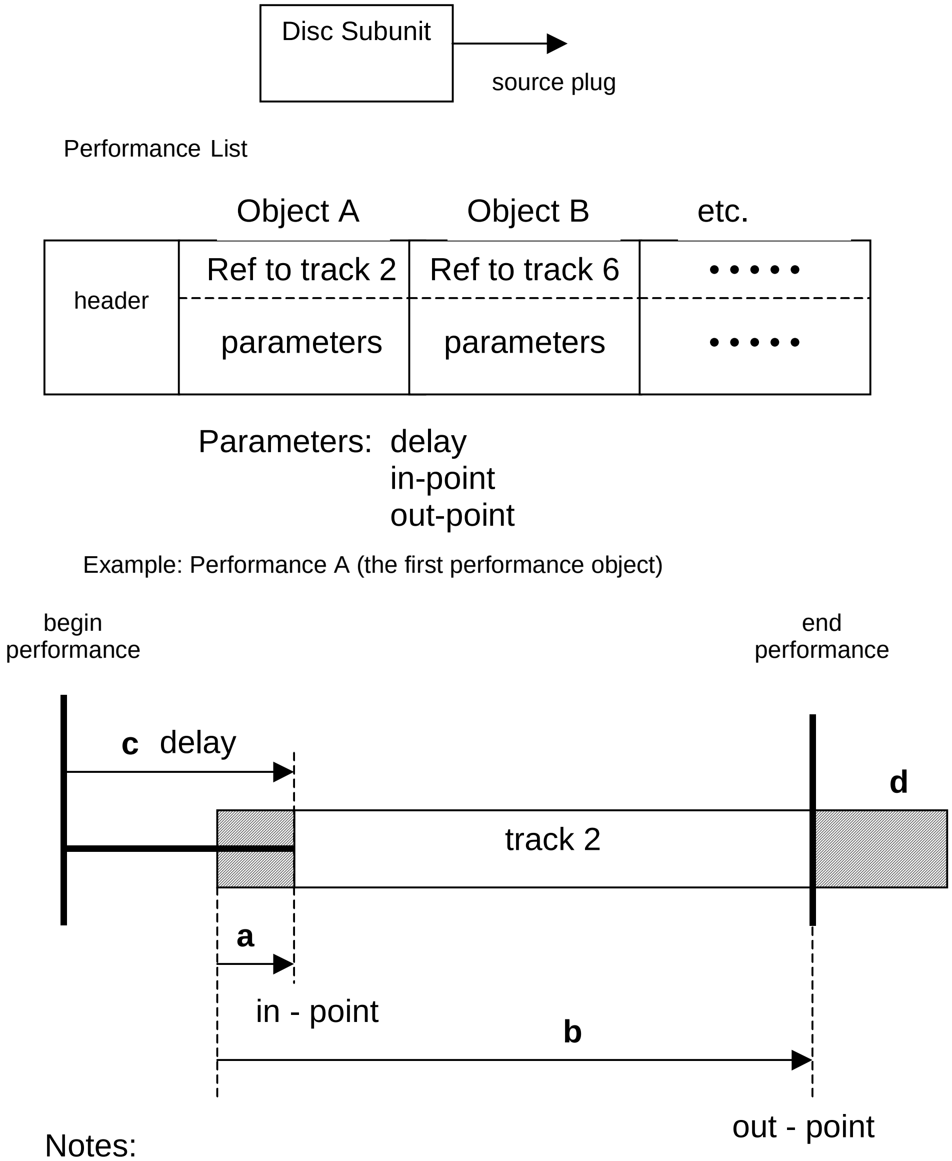
the start time of the item, relative to the start time of the performance

the “in-point”, or offset from the beginning of the item, at which the performance of that item begins

the “out-point”, also measured as an offset from the beginning of the item, at which the performance of that item ends

Note that for items which do not have the concept of duration, specifically the digital still image and text objects, the in-point and out-point attributes have no meaning. The entire item is “performed”, or transmitted, at the specified start time. The amount of time actually required to transmit the bytes is not involved in the performance specification.

The following diagram illustrates an example performance:



Notes:

a) in-point is measured from the beginning of track 2
b) out-point is measured from the beginning of track 2
c) delay is measured from the beginning of the performance
d) the areas with hash marks are “trimmed out”, and are not played. Note that the performance ends at the out-point.

Figure 9-6 example of performance

The following diagram enhances the example to show how a child list can be used to play several items ***sequentially*** (not ***simultaneously***) in a single performance.

> **NOTE:** The way a performance is carried out differs between main and child performance lists. In the main performance list, the execution of object n begins immediately after object n-1 completes (so, object n starts playing relative to object n-1). However, in child performance lists, the delay time is relative to the beginning of that child list’s performance time.

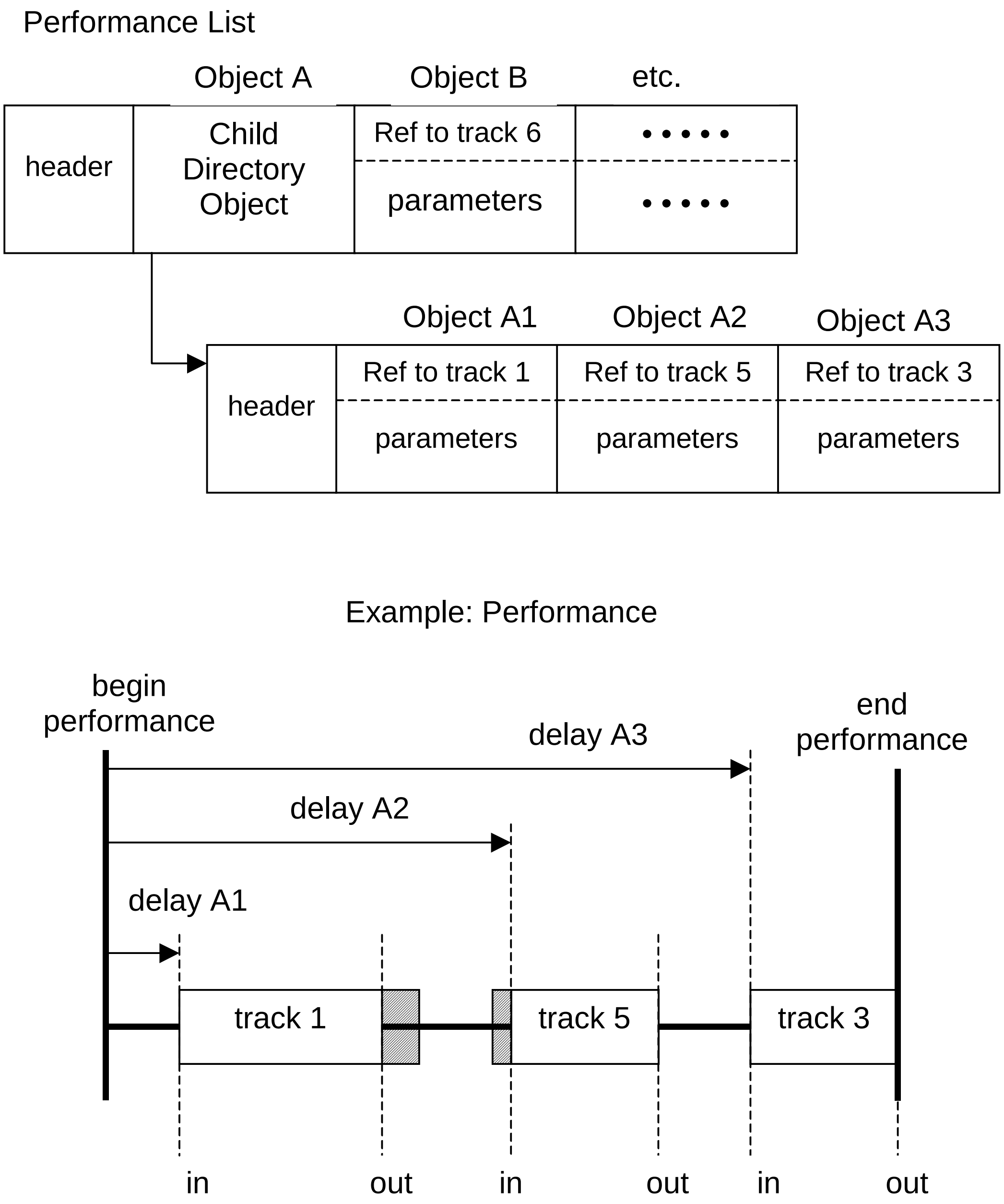


Figure 9-7 example of performance with child list

9.6.2 Default vs. User-Modifiable Performance Lists

The contents of performance lists may be user-modifiable, or their contents may be determined by the subunit implementation. For example, a particular media format might have a performance list which is forced to mirror all or part of the contents on the media;

changes to the contents lists are reflected by changes in these performance lists. Such performance lists are referred to as Default Performance Lists.

> **NOTE:** Default performance lists are just a concept; there is no special list data structure with a list_type of default_performance_list.

A subunit implementation may choose to keep a set of user-modifiable performance lists, which allow the creation of custom performances. The disc subunit media type specifications will indicate if such lists are possible for a given media type, and what restrictions, if any, are place on these lists.

A subunit may choose to keep both sets of performance lists – default and user-modifiable.

9.6.3 Performance List Hierarchy Structure

Supporting performance lists is optional. If they are supported, then the subunit implementation decides how many performance lists will be implemented. There is NOT a fixed, one-to-one relationship between a given performance list and a given source plug.

In the subunit identifier descriptor shall be the ID of a **Root Performance List**.

All other performance lists shall be main or children under the root performance lists as shown in the example diagrams.

Root performance lists contain child directory objects. If the list contains child directory objects, then Each of them points to a **Main Performance List**.

Main performance lists contain either performance objects or child directory objects. If the list contains child directory objects, then each of them points to a **Child Performance List**.

All of the objects in a given performance list shall refer to the same type of content (all audio track references, all digital still image references, etc.).

There may be any number of object entries allowed in the performance lists. The number of entries may be pre-allocated (i.e. a list always has 10 object entries, even if some of them are empty), or they may be added incrementally as new objects are created. This is an implementation choice.

The following diagram illustrates the hierarchy rules:

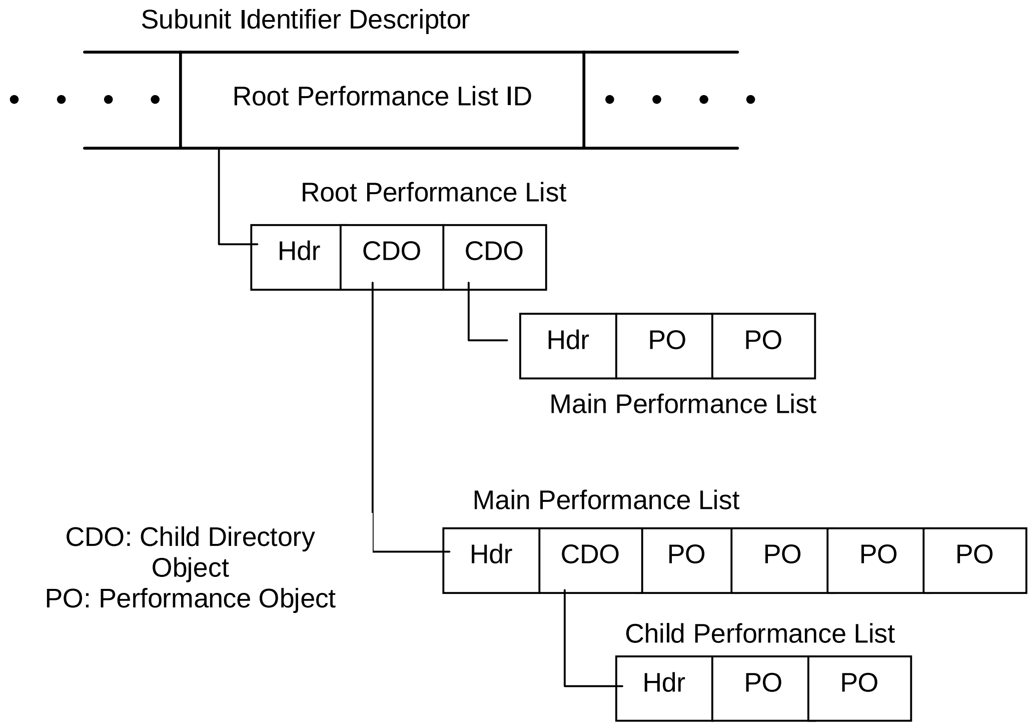


Figure 9-8 hierarchy rules for performance list

9.6.4 Performance List list_specific_information

The performance list contains some number of performance objects (see section 8.6). The *list_specific_information* field is as follows:

Performance List list_specific_information	
Address Offset	Contents
00 00$_{16}$	non_info_block_fields_length
00 01$_{16}$	
00 02$_{16}$	disc_subunit_list_attributes
00 03$_{16}$	AV_object_type
00 04$_{16}$	time_stamp_info_block (descriptor modification time)
:	
:	
:	size_indicator_info_block (total duration of all performances in list)
:	
:	
:	other optional info blocks
:	
:	

Table 9-10 Performance List list_specific_information

The *disc_subunit_list_attributes* field is as described above for the common disc subunit list attributes.

The *AV_object_type* field specifies the type of AV objects that are referenced by the performance objects in this list (remember, all performance objects in a given performance list must refer to the same type of AV object). This field is encoded as defined in section 8.1.

The *time_stamp_info_block* indicates when this list was list changed. This info block is required.

The *size_indicator_info_block* indicates the total clock time duration (hours:minutes:seconds:x10ms) of all performances in this list.

Other optional info blocks which may be interesting to include are the name and image info blocks; many others could be used as well. Controllers should be prepared to find any number of any type of info blocks at any time, and not treat this as an error.

9.7 Synchronized Performance Lists (Synchro Lists)

9.7.1 Overview of Synchronized Performances

The previous sections described the basic concept of a performance, which allows one source plug to play several content items sequentially. The items are played sequentially on a given source plug, and several different plugs may play simultaneously.

It is possible to perform such a sequence of objects not only on one source plug of a subunit, but also on several source plugs simultaneously. Such a performance involves many performance lists, one for each of the source plugs used for the performance. When two or more performance lists are played simultaneously on two or more source plugs, this is called a **synchronized performance**. A synchronized performance is a way of specifying which content items are to be played, and the order in which they should be played. It allows the performance list collection to be used in a very flexible manner.

A **synchronized performance object** is used to specify which performance, indicated by a position value, should be played.

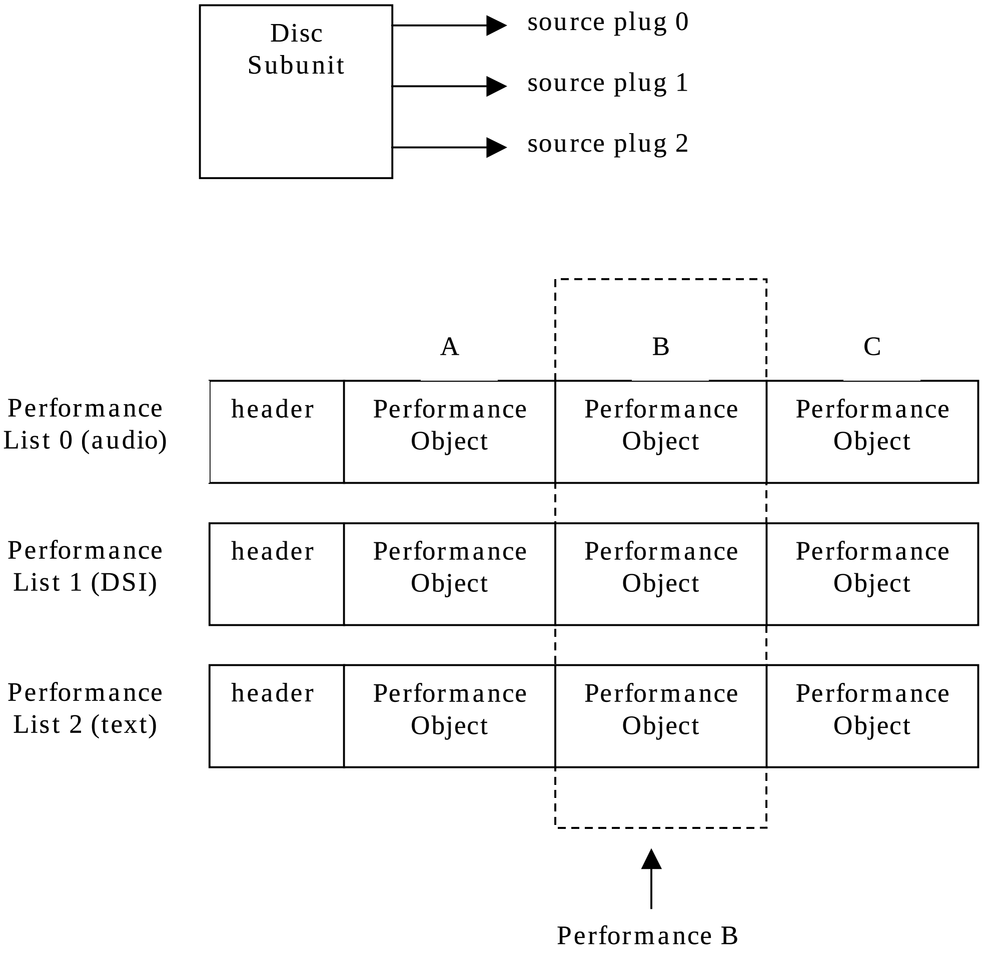
To facilitate synchronized performances; the **Synchronized Performance List** (or **Synchro List**) structure is defined. It is responsible for synchronizing the playback of two or more Performance Lists (described earlier).

A synchronized performance list specifies which set of performance lists are used in a synchronized performance, and the source plugs which they will be played on. This is why there is no fixed relationship between subunit source plugs and performance lists; the relationship is defined dynamically, when a synchronized performance is played. A synchronized performance may use a subset of the existing performance lists.

A synchronized performance may also specify contents lists. In this situation, contents lists are treated as performance lists in which “there is no in-point, out-point or delay time specification” for the content objects to be performed.

The following diagrams illustrate the performance of several items simultaneously, and the use of synchronized performance lists to customize the order of performances.

To understand how synchronized performances work, it is useful to imagine all of the performance lists drawn one above the other, so that the lists and each of their objects form a table or matrix. Each “column” of the table represents a performnce, as illustrated below:



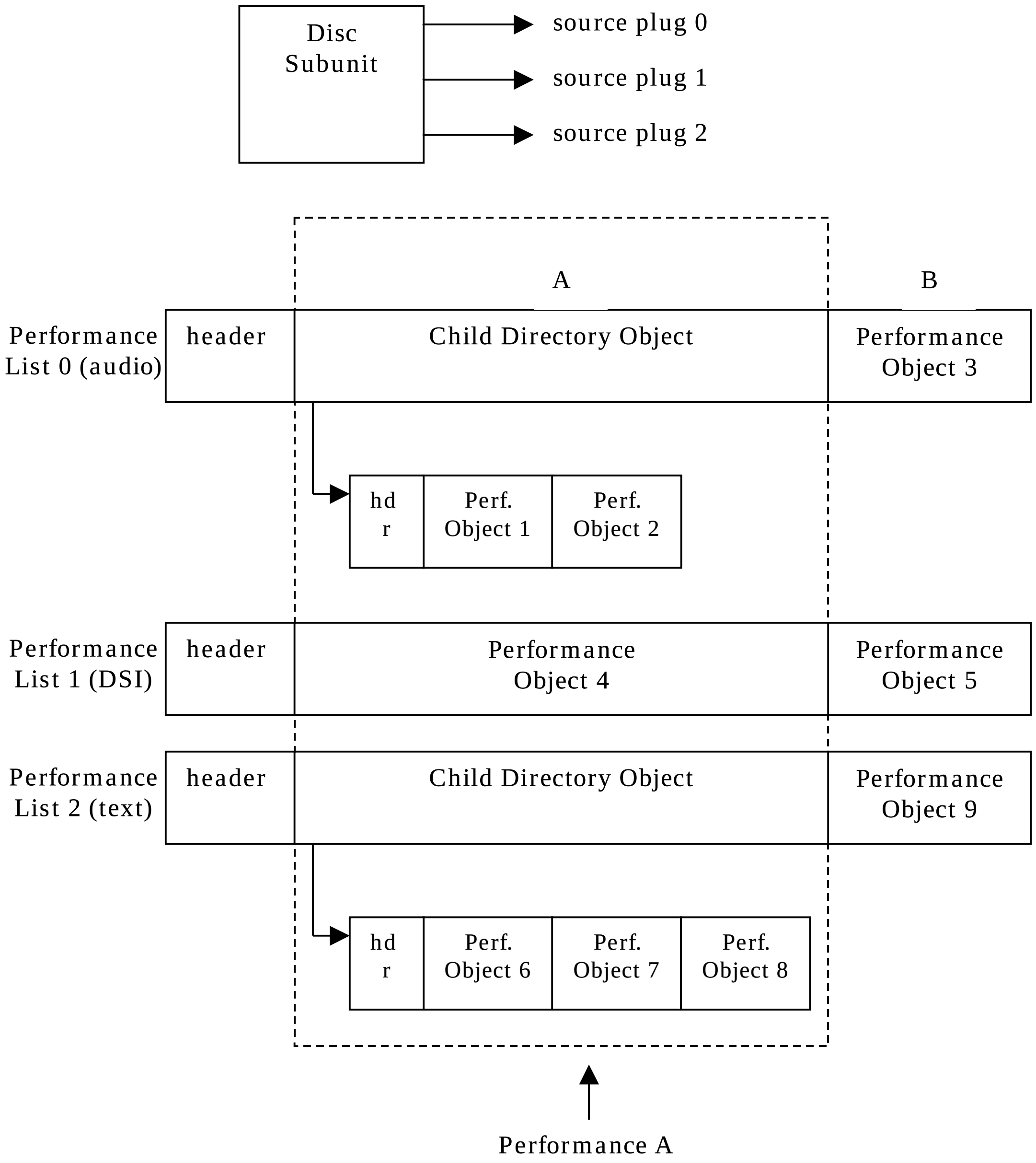
The content items represented by the objects in column B make up “performance B”. When performance B is triggered, all three streams of content will be played at the same time.

NOTE: A general rule for all performance lists: the objects in a list must all refer to the same TYPE of object (DSI, audio track, etc.). A synchronized performance can include lists with different types of objects, as shown in the example above.

Figure 9-9 synchronized performances

NOTE: There is no fixed relationship between the number of subunit source plugs and the number of performance lists a subunit can support. This is a product implementation issue.

As described previously, it is possible for one of the source plugs to transmit several different content items during a single performance, using a child list. This is also valid in the synchronized performance:



Performance A consists of the following:
- performance objects 1 and 2 on source plug 0
- performance object 4 on source plug 1
- performance objects 6, 7 and 8 on source plug 2

The streams on plugs 0, 1 and 2 are played at the same time; the objects on a given plug are played sequentially.

Figure 9-10 synchronized performances with child lists

The synchronized performance list structure, along with its synchronized performance objects, is the means of encapsulating all of the above “multi-stream” performance information. It also allows the customization of the order in which performances are played, so that they do not have to be in the order of the performance objects in performance lists. This is shown in the following diagram:

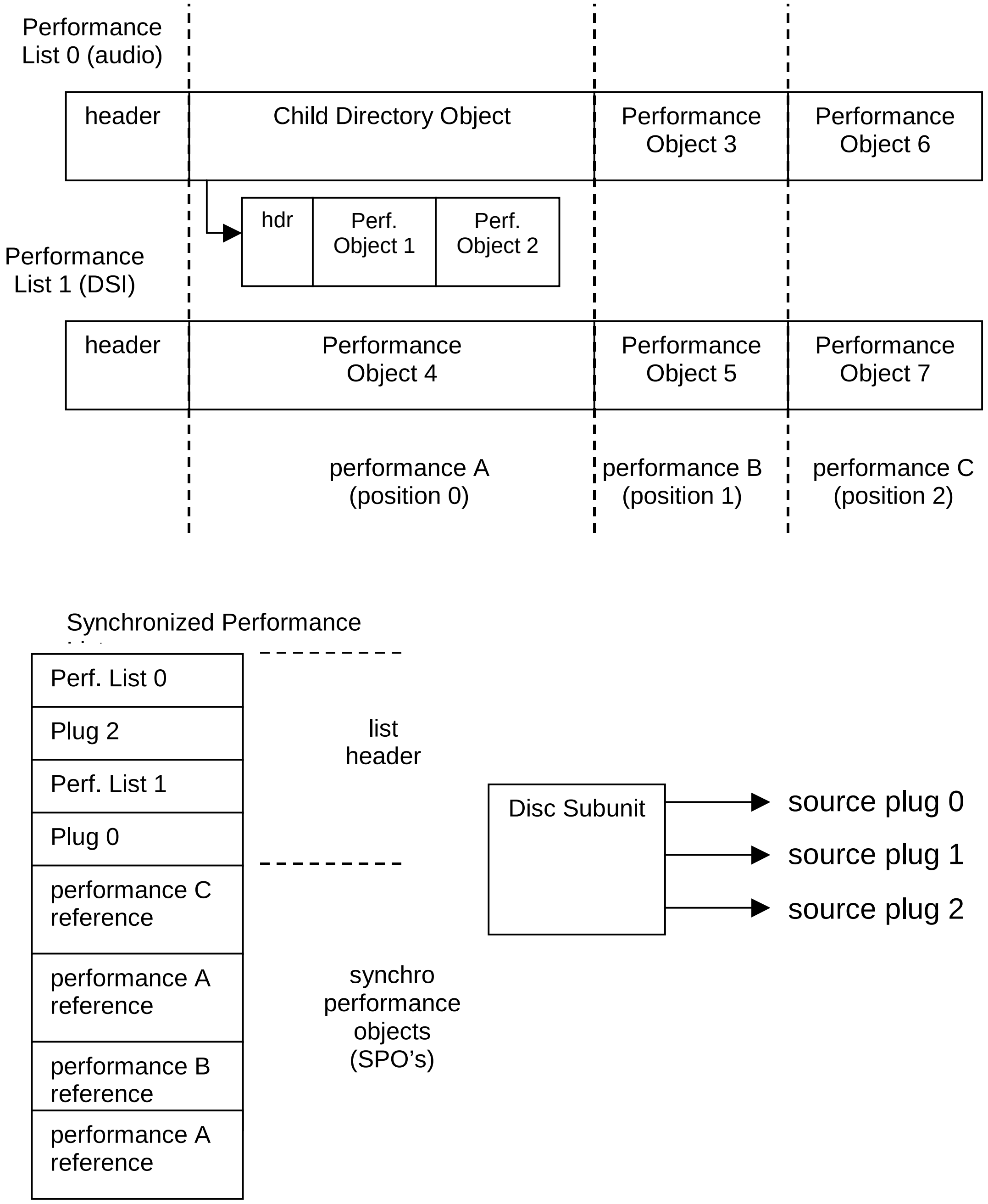


Figure 9-11 synchronized performance list structure

In the diagram, the synchro performance list header contains a *synchro_performance_list_and_plug_pairs_info_block* that associates performance list 0 with subunit source plug 2, and performance list 1 with subunit source plug 0. In this example, subunit source plug 1 is not used. The list contains several synchro performance objects (SPO's), each of which refers to a performance (by the position of the performance "column" in the matrix). There are other fields and info blocks of the list header not shown in this example diagram.

The synchro performance objects will be played sequentially from the list. Therefore, the entire synchronized performance will consist of the content objects for performance C, followed by performance A, followed by performance B, and then performance A again. Thus, the order of performance has been customized by the placement of the synchro performance objects in the synchro performance list.

In all synchronized performances, the content items from a specified performance (or "column in the matrix") are played together. The duration of any given performance is determined by the longest duration of its component elements. In the example diagram above, if the performance component represented by A0 (performance list 0, performance column A) had a total duration of 5 minutes (total time needed to play objects 1 and 2), and performance component A1 had a total duration of 2 minutes (for object 4), then the total duration of performance A would be 5 minutes.

9.7.2 Synchronized Performance Plug Group (Synchro Plug Group)

Synchronized performance plug group (also called synchro plug group) are defined to support the AV/C subunit plug model and the perfomance of several simultaneous streams of data.

The synchro plug group can be used in some performance-related commands (such as Play, etc.) where regular subunit source plugs can be used. Synchro plug group can not be used in all commands; for example, it is not possible to use CONNECT with synchro plug group.

Synchro plug group can be used for controlling and monitoring synchronized performances. Supporting these types of plugs is an optional implementation feature. The following diagram illustrates the conceptual model of synchro plug group:

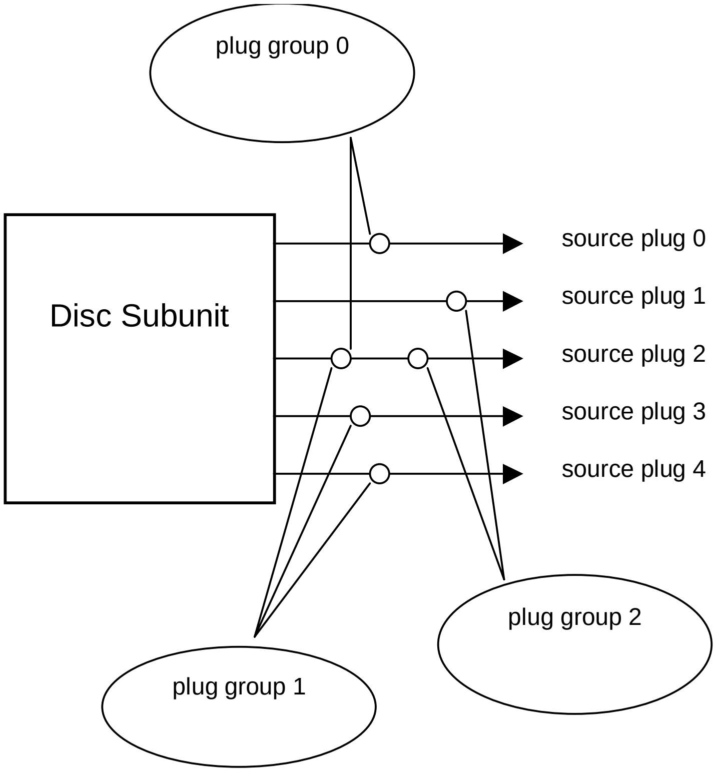


Figure 9-12 conceptual model of synchro plug group

As shown, synchro plug group 0 is associated with subunit source plugs 0 and 2; synchro plug group 1 represents subunit source plugs 2, 3 and 4; and synchro plug group 2 represents subunit source plugs 1 and 2.

Synchro plug group data structures can be found in the disc subunit status descriptor structure; the subunit shall report as many synchro plug group as the number of simultaneous streams it can handle.

9.7.3 Synchronized Performance List Hierarchy

There may be many synchronized performance lists. As with the performance lists, in the subunit identifier descriptor is the ID of a master synchronized performance list. The master list may contain a collection of synchronized performance objects (if it is the only synchronized performance list), or it may contains a set of child directory objects (each pointing to a synchronized performance list).

There is two levels to this hierarchy.

The following diagram illustrates these rules:

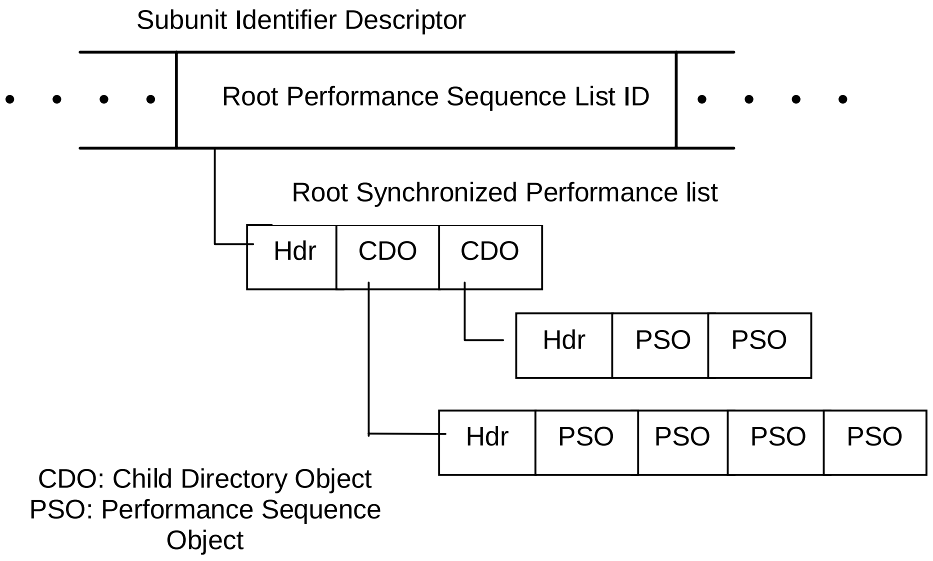

Subunit Identifier Descriptor
Root Performance Sequence List ID
Root Synchronized Performance list
Hdr
CDO
CDO
Hdr
PSO
PSO
Hdr
PSO
PSO
PSO
PSO
CDO: Child Directory Object
PSO: Performance Sequence Object

Figure 9-13 synchronized performance list hierarchy

9.7.4 Synchronized Performance List list_specific_information

A synchronized performance list contains several synchronized performance objects (see section 8.7). The *list_specific_information* specifies which performance lists are to be used for ALL of the performances represented by this list, and for each list, the subunit source plug to be used. Additionally, it contains a time stamp for when the list was last updated, and possibly other info blocks such as the name of the list, etc.

The following diagram illustrates the *list_specific_information* field of the synchronized performance list:

Synchronized Performance List list_specific_information	
Address Offset	Contents
00 00$_{16}$	non_info_block_fields_length
00 01$_{16}$	
00 02$_{16}$	disc_subunit_list_attributes
:	synchro_performance_list_and_plug_pairs_info_block
:	
:	
:	time_stamp_info_block (descriptor modification time)
:	
:	
:	other optional info blocks
:	
:	

Table 9-11 Synchronized Performance List list_specific_information

The *disc_subunit_list_attributes* field is as described above for the common disc subunit list attributes.

The *synchro_performance_list_and_plug_pairs_info_block* specifies the {list ID, source plug} pairs to be used for the performance. This info block is required except for root synchronized performance sequence list.

The *time_stamp_info_block* indicates when this list was list changed. This info block is required.

Other optional info blocks which may be interesting to include are the name and image info blocks; many others could be used as well. Controllers should be prepared to find any number of any type of info blocks at any time, and not treat this as an error.

9.7.5 Establishing and Controlling Synchronized Performances

Generally, setting up and controlling synchronized performances is very similar to controlling the playback of a single stream of data.

The synchro list list_specific_information contains a *synchro_performance_list_and_plug_pairs_info_block* structure. This info block contains sets of {list_id, source plug_id} pairs that form the collection of lists and subunit source plugs for that synchro performance. The controller can use the WRITE INFO BLOCK command to change this info block, and therefore the associations between a given synchro list and the real subunit source plugs and individual performance lists.

The controller must use the ASSOCIATE LIST WITH PLUG command to associate a specified synchro list with a specified synchro plug group. The controller can examine the *position_info_block*, nested inside of the *plug_status_info_block* of the synchro plug group, in the disc subunit status descriptor to find out which synchro list it has been associated with. Refer to the synchro plug group info block data structures for details.

For runtime management of synchronized performances, the controller can use commands that take subunit source plugs as parameters. For example, the PLAY command, specifying the synchro plug group, will trigger the synchronized performance.

The following rules apply to the management of synchronized perfances:

1) During a synchronized playback operation, commands that specify synchro plug group shall be accepted.

2) For each real subunit source plug, commands that will have an affect on the synchronized playback operation shall be rejected:

a) Operation commands such as Play, Stop, Search, etc. shall be REJECTED

b) Configuration commands that change the playback order and variable speed shall be REJECTED

c) Configuration commands to change mute and variable pitch may be ACCEPTED.

3) When the PLAY command is accepted for a synchro plug group, the configuration and list ID of the corresponding source plugs are overwritten at that time.

9.7.6 Monitoring Synchronized Performance Status

The status of synchronized performances can be monitored via the disc subunit status descriptor mechanism. For details, refer to section 7.

9.8 Text Database Lists

9.8.1 Overview of the Text Database Concept

Several items in the disc subunit model have some kind of text field(s) related to them. Examples include the titles of audio tracks and the disc, list titles, etc. In many cases, these text fields may exist in several different languages.

In order to optimize the overall descriptor mechanism, the text database concept was defined. This allows all text fields to be stored as objects in one or more lists, and to be referred to from those objects or other descriptors that need them. Thus, the variable-length text fields are not embedded in the middle of other descriptors, which at times may complicate the descriptor structure management process.

The storage strategy for these text database objects might be mandated by a particular disc subunit media type specification. In the absence of a media-type-specific method, the subunit vendor is free to store the text database objects in any manner desired.

Examples of storage strategies might include the following:

Define a separate list for each text item (the title of track 1, the title of track 2, etc.). In this list can be several text database objects, each one specifying that text item in a different language (English, Japanese, French, etc.). There may or may not be any relationship in the ordering of items across the lists (each list is independent, and text database objects in a given language may be in different positions in each list).

Define a separate list for each language supported by the subunit or the media type (English, Japanese, French, etc.). In each list can be several text database objects, each one specifying a text item (title of track 1, title of track 2, etc.). As with item “a”, the ordering of objects is not necessarily consistent across lists.

Define a single list containing ALL of the text database objects, for all text items, in all supported languages.

Any other possible combination…

There is no particular limit to the number of text database lists supported by a subunit, other than practical considerations.

When a descriptor needs to include one or more text fields, it can refer to the entries in the text database list. Depending on how the text database is implemented, the method of referencing text database objects involves one of the following:

Point to a list. In this case, the controller who is parsing the descriptor can assume that all entries in the list represent the same text item (such as the title of track 1), each in a different language.

Point to each text database object which represents the specified text item. In this case, the text database objects for each language are stored in a manner which requires individual references.

Include the text directly in the descriptor structure (immediate storage). This method does not use the text database mechanism.

In the current text database model, objects are created and added to/removed from the database automatically. The conditions in which these actions occur are specified by the disc subunit media type. For details, please refer to the appropriate disc subunit media type specification.

9.8.2 Text Database List Hierarchy Structure

Supporting text database lists is optional. If supported, there shall be one text database list as a root list, whose ID value is found in the disc subunit identifier descriptor. If more than one list is used, then this list shall contain one or more child directory objects, each of which points to a text dbase list. The root list may also contain text database objects.

There is no pre-defined limit to the number of levels in the text database list hierarchy, but some disc subunit media type specifications may impose strict limits on the structure of the database hierarchy. For details, please refer to the appropriate media type specification.

9.8.3 Text Database Object List list_specific_information

The text database list contains text database objects (see section 8.8). The *list_specific_information* field has the following format:

Text Database List list_specific_information	
Address Offset	Contents
00 00$_{16}$	non_info_block_fields_length
00 01$_{16}$	
00 02$_{16}$	disc_subunit_list_attributes
00 03$_{16}$	time_stamp_info_block (descriptor modification time)
	other optional info blocks

Table 9-12 Text Database List list_specific_information

The *disc_subunit_list_attributes* field is as described above for the common disc subunit list attributes.

The *time_stamp_info_block* indicates when this list was list changed. This info block is required.

Other optional info blocks which may be interesting to include are the name and image info blocks; many others could be used as well. Controllers should be prepared to find any number of any type of info blocks at any time, and not treat this as an error.

10. Disc Subunit Commands

		defined *ctypes*			
Opcode	Value	C	S	N	Comments
ACCEPT/REJECT EDITING CHANGES	$D2_{16}$	X	–	–	Commit editing changes to the disc media, or reject in-progress editing changes.
ASSOCIATE LIST WITH PLUG	$D3_{16}$	X	–	–	Associate a list with a source or destination plug
AUTO UPDATE ON/OFF	$D4_{16}$	X	–	–	Enable or disable automatic updates of editing changes
COMBINE	41_{16}	X	–	–	Combine two tracks into a single track
CONFIGURE	$D1_{16}$	X	–	–	Prepare the subunit for a recording or playback operation
DISC STATUS	$D0_{16}$	–	–	X	Request notification when the status of the disc subunit changes
DIVIDE	42_{16}	X	–	–	Separate a specified track into two tracks
ERASE	40_{16}	X	–	–	Erase the entire AV contents of the disc subunit, just a specified track, or a specified portion of the disc storage space
IMPORT/EXPORT MEDIUM	$C1_{16}$	X	–	–	Put the disc into or remove it from the drive mechanism
MONITOR	$C6_{16}$	X	–	–	Listen to what is being recorded
MOVE	43_{16}	X	–	–	Move a track to a different logical location (assign a new object position number)
INCREMENT OBJECT POSITION NUMBER	51_{16}	X	–	–	Divide a track while recording
OBJECT NUMBER SELECT	$0D_{16}$	X	–	–	Select one or more objects for transmission, (response after completion)
PLAY	$C3_{16}$	X	–	–	Begin playing the disc (immediate response)
RECORD	$C2_{16}$	X	–	–	Record a streaming object (audio track, etc.)
RECORD OBJECT	56_{16}	X	–	–	Record a non-streaming object (still image, etc.)
REHEARSAL	$C7_{16}$	X			Playback a few positions continuously
SEARCH	50_{16}	X	–	–	Perform a relative or absolute search for the specified location on the media
STOP	$C5_{16}$	X	–	–	Stop the current operation
UNDO	44_{16}	X	–	–	Undo the most recent editing operation(s)

Table 10-1 ctype definition for Disc Subunit Command

NOTE: Profiles will specify the command support requirements. Profiles are defined in disc subunit media-type-specific documents. The table presented here only specifies which *ctypes* are defined for the indicated commands (“X” = defined, “–“ = undefined).

10.1 Comments about the Editing Commands

Several of the commands defined for the disc subunit involve editing content on the media. Currently, editing commands include {DIVIDE, COMBINE, MOVE, ERASE and UNDO}. All of these commands involve the contents lists (described in sections 9.3 and 9.4). Modifications to the contents lists will affect the content on the media.

When other lists are modified, the media contents are not affected.

10.2 Disc Subunit Command Frame Structure

In order to allow the most efficient command processing in controllers and subunit implementations, the command frames for the disc subunit commands have been optimized. This section describes the new command frame format and how it is represented in this specification.

All of the commands now fall into one of three categories (described below). Commands from a given category can be described by their “command frame parts”.

Note that the command frame structures described here apply only to the commands defined specifically for the disc subunit. Other existing AV/C commands, such as common unit and subunit commands which a disc subunit might be required or choose to implement, are not covered by these sections.

10.2.1 Common Command Frame Header

All of the commands defined for the disc subunit specification have the same common “command header” in the command frame. This header is shown in the figure below:

	msb							lsb
opcode	opcode							
operand[0]	result							
operand[1]	subfunction_1							
operand[2]	subfunction_2							

Table 10-2 Common Command Frame Header

The *opcode* contains the opcode for the command.

The *result* field shall be set to FF$_{16}$ by the controller in the command frame. In the response frame, the subunit shall update the field with a result code. Result codes and their interpretation are defined for each command.

The *subfunction_1* and *subfunction_2* fields are defined by each command, if necessary. If a particular command does not need these subfunctions, then they shall be treated as reserved, set to 00$_{16}$ values, as specified in section 4.6 Rules for Reserved Fields.

These four bytes shall be referred to in command frame descriptions as the *common_command_header_part*.

All commands shall have exactly ONE *common_command_header_part* in their command frames.

10.2.2 Other Command Frame Parts

Each of the disc subunit commands falls into one of three categories:

category A: commands that affect subunit plugs (also called *control* commands)

category B: commands that affect content on the media (also called *editing* commands)

category C: "other" commands (miscellaneous commands)

These command categories have nothing to do with the functionality of the command, or the AV/C command type (ctype). These categories are ONLY used to define further command frame parts in addition to the common header. All of the commands that fall into a particular category will share these additional common command frame parts. Some parts are optional and may not exist in a given command.

The following table defines the command frame parts for each of these categories:

command category	command frame parts	meaning
A – commands that affect subunit plugs	descriptor_identifier_part	Contains a descriptor_identifier structure, as defined in reference[1].
	plug_identifier_part	Describes a subunit plug (source, destination or synchro plug group).
	control_position_indicator_part	Specifies a position for the control operation.
	control_range_specification_part	Contains two control_position_indicator_part structures, denoting an "in point" and an "out point".
B – commands that affect content on the media	descriptor_identifier_part	This is the same as the category A part.
	edit_location_indicator_part	Specifies editing position information.
	edit_range_specification_part	Contains two edit_location_indicator_part structures, denoting an "in point" and an "out point".
C – miscellaneous commands	none	There are no common parts defined for the miscellaneous command category. Each of these commands is defined independently.

Table 10-3 Command Frame Parts

Generally, there are no restrictions on how many parts, or how many instances of a given part, may be defined for a command frame. A command in a given category may have from zero to n of the parts defined for that category. Each command description will detail the specific parts that it requires.

10.2.2.1 The descriptor_identifier_part

For disc subunit command frames, the *descriptor_identifier_part* is used to specify objects or lists. The format of the *descriptor_identifier_part* is the same as the standard *descriptor_identifier* in reference[1]:

	msb							lsb
operand[n]	descriptor_type							
:	descriptor_type_specific_reference							
:								
:								

Table 10-4 the *descriptor_identifier_part* format

The *descriptr_type* and *descriptor_type_specific_reference* fields are described in the OPEN DESCRIPTOR command of reference[1].

The *descriptor_type* values that are used by the disc subunit commands are as follows:

descriptor_type	meaning
10_{16}	Object list descriptor - specified by list ID
11_{16}	Object list descriptor - specified by list_type
20_{16}	Object entry descriptor - specified by object position
21_{16}	Object entry descriptor - specified by an object ID

Table 10-5 descriptor_type

Note that the above values are a subset of the full range of descriptor types defined by the AV/C general specification.

10.2.2.2 The plug_identifier_part

The *plug_identifier_part* is used to specify a subunit plug to be used by the command. It has the following format:

	msb							lsb
operand[n]	plug_type							
operand[n + 1]	plug_id							

Table 10-6 *plug_identifier_part* format

The *plug_type* field specifes either a source (= 0), destination (= 1) or synchro plug group (= 2). All other values for this field are reserved for future specification.

The *plug_id* contains the ID of the plug being specified.

10.2.2.3 The control_position_indicator_part

The *control_position_indicator_part* contains the *indicator_type* and *indicator_type_specific* fields of a *position_indicator_info_block*, as specified in reference[1]:

	msb							lsb
operand[n]	indicator_type							
:	indicator_type_specific							
:								
:								

Table 10-7 *control_position_indicator_part* format

The full set of position indicator type values as defined in reference[1] are valid for use in the disc subunit command frames.

> IMPORTANT: For category A commands (e.g. *control* commands), it is important to understand the relationship between the specified plug(s), the list and object(s) used by the command. The command frames do not have a list specification in them, but they do have a plug specification. The command ASSOCIATE LIST WITH PLUG is used to associate a given list with a given plug. When the category A commands have object position-relative indicators, it is implied that the object position is in the list that is associated with the plug specified in the command frame.

10.2.2.4 The control_range_specification_part

The *control_range_specification_part* defines a range of content on the media (an in-point and an out-point to use for the command). It is composed of two *position_indicator_part* structures:

	msb							lsb
operand[n]	position_indicator_part (in-point)							
:								
:								
:	position_indicator_part (out-point)							
:								
:								

Table 10-8 *control_range_specification_part* format

10.2.2.5 The edit_location_indicator_part

The *edit_location_indicator_part* specifies where an editing operation is to take place. This includes a descriptor and a position in the content referred to by that descriptor, for the operation. The following diagram shows its format:

	msb							lsb
operand[n]	descriptor_type							
:	descriptor_type_specific							
:								
:								
:	indicator_type							
:	indicator_type_specific							
:								
:								

Table 10-9 *edit_location_indicator_part* format

The *descriptor_type* and descriptor_type_specific have the same meaning and use as those described for the *descriptor_identifier_part*.

The *indicator_type* and *indicator_type_specific* fields are similar to, but NOT the same as, those for the *control_position_indicator* part described above. Specifically, there are two differences:

there are only three indicator types supported

in those indicator types, the object position reference has been omitted

The *indicator_type* field can take on the following values (note that these values are not the same as for the *control_position_indicator_part* equivalent fields):

indicator_type	meaning
00$_{16}$	relative_HMSF_count
01$_{16}$	relative_segment_count
02$_{16}$	relative_byte_count
03$_{16}$ - FF$_{16}$	reserved for future specification

Table 10-10 *indicator_type*

The *indicator_type_specific* fields for each of the indicator types are described below:

indicator_type = 00$_{16}$ (relative_HMSF_count)		
:	indicator_type = 00$_{16}$	
:	+ / -	hours
:	minutes	
:	seconds	
:	frames	

Table 10-11 indicator_type = 00$_{16}$ (relative_HMSF_count)

All of the fields have the same meaning and use as for the standard relative HMSF count structure (reference[1]).

indicator_type = 01$_{16}$ (relative_segment_count)	
:	indicator_type = 01$_{16}$
:	segment_number
:	

Table 10-12 indicator_type = 01$_{16}$ (relative_segment_count)

The *segment_number* field has the same meaning and interpretation as for the object position-relative segment count (reference[1]).

indicator_type = 02$_{16}$ (relative_byte_count)	
:	indicator_type = 02$_{16}$
:	length_of_byte_offset
:	byte_offset
:	
:	

Table 10-13 indicator_type = 02$_{16}$ (relative_byte_count)

The *length_of_byte_offset* and *byte_offset* fields have the same meaning and interpretation as for the object position-relative byte count (reference[1]).

IMPORTANT: For category B commands (e.g. *editing* commands), there are two ways to specify the location of the operation.

If the descriptor_type indicates a list, then the indicator_type shall be an “in list” offset indicator, as described above in section 10.2.2.3 The control_position_indicator_part.

If the descriptor_type indicates an object, then the indicator_type shall be as described for the “in object” offset indicator in this section.

10.2.2.6 The edit_range_specification_part

The *edit_range_specification_part* defines a range of content on the media (an in-point and an out-point to use for the command). It is composed of two *edit_location_indicator_part* structures:

	msb							lsb
operand[n]	edit_location_indicator_part (in-point)							
:								
:								
:	edit_location_indicator_part (out-point)							
:								
:								

Table 10-14 *edit_range_specification_part*

10.2.2.7 Category A Commands

The following table contains the commands that fall into category A (commands that affect the subunit plugs). For each command, it shows which parts are Mandatory or Optional. Mandatory parts are always used; Optional parts appear based on the subfunctions:

		parts used by each command				
Command	Opcode	plug	pos	range	descriptor	specific
ACCEPT/REJECT EDITING CHANGES	D2$_{16}$					
ASSOCIATE LIST WITH PLUG	D3$_{16}$	M			O	
AUTO UPDATE ON/OFF	D4$_{16}$					
CONFIGURE	D1$_{16}$	M			O	
IMPORT/EXPORT MEDIUM	C1$_{16}$					
MONITOR	C6$_{16}$	M				
INCREMENT OBJECT POSITION NUMBER	51$_{16}$	M				
PLAY	C3$_{16}$	M				
RECORD	C2$_{16}$	M	O			M
RECORD OBJECT	56$_{16}$	M			M	M
REHEARSAL	C7$_{16}$	M		M		
SEARCH	50$_{16}$	M	O			O
STOP	C5$_{16}$	M				

Table 10-15 Category A Commands

The general structure of category A commands is as follows:

	msb							lsb	
opcode	OPCODE (XX_{16})								*common_command_header_part*
operand[0]	result								
operand[1]	subfunction_1								
operand[2]	subfunction_2								
operand[3]	plug								*plug_identifier_part*
operand[4]									
operand[5]	opcode_specification								*plug_identifier_part /*
:									*descriptor_identifier_part /*
:									*control_position_indicator_part /*
:									*control_range_specification_part /*
:									*opcode_specific_operands*

Table 10-16 The general structure of category A commands

All category A commands have the common header part. If the command uses a plug specification, it appears next. Subsequent parts, if needed, appear after the plug specification. The exact set of parts used is dependent on the command.

10.2.2.8 Category B Commands

The following table contains the commands that fall into category B (commands that affect the subunit in general). For each command, it shows which parts are Mandatory or Optional. Mandatory parts are always used; Optional parts appear based on the subfunctions:

		parts used by each command			
Command	Opcode	pos	range	descriptor	specific
COMBINE	41_{16}			M	
DIVIDE	42_{16}	M			
ERASE	40_{16}		O	O	
MOVE	43_{16}			M	
UNDO	44_{16}				

Table 10-17 Category B Commands

The general structure of category B commands is as follows:

	msb							lsb	
opcode	OPCODE (XX_{16})								*common_command_header_part*
operand[0]	result								
operand[1]	subfunction_1								
operand[2]	subfunction_2								
operand[3]	opcode_specification								*descriptor_identifier_part /*
:									*edit_location_indicator_part /*
:									*edit_range_specification_part /*
:									*opcode_specific_operands*

Table 10-18 The general structure of category B commands

All category B commands have the common header part. Subsequent parts, if needed, appear after the common header part. The exact set of parts used is dependent on the command.

10.2.2.9 Category C Commands

The following table contains the commands that fall into category C (miscellaneous commands):

Opcode	Value
CREATE DESCRIPTOR	$0C_{16}$
DISC STATUS	$D0_{16}$
OBJECT NUMBER SELECT	$0D_{16}$
OPEN INFO BLOCK	05_{16}
READ INFO BLOCK	06_{16}
WRITE INFO BLOCK	07_{16}

Table 10-19 Category C Commands

Note that of the commands in this category, only DISC STATUS is a disc subunit-specific command; the others are general AV/C commands that most disc subunits would probably implement, which is why they are mentioned here.

Each of the commands in category C has its own format, so there is no standard structure.

10.2.3 Opcode-Specific Part

The *opcode_specific_part* contains those fields that are unique to a given command. These will be detailed in the description of each command. This part appears at the end of the command frame, after all of the standard parts for that command.

10.3 ACCEPT/REJECT EDITING CHANGES

The ACCEPT/REJECT EDITING CHANGES command is used to either permanently commit editing changes the media, or to reject all outstanding editing changes. When accepted, the command will copy the editing changes from the temporary contents list to the disc media. For more details on editing, please refer to section 9.5. The control command has the following format:

	msb							lsb	
opcode	ACCEPT/REJECT EDITING CHANGES ($D2_{16}$)								*common command header part*
operand[0]	result								
operand[1]	subfunction_1								
operand[2]	reserved								

Table 10-20 ACCEPT/REJECT EDITING CHANGES command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00_{16}	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF_{16}	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-21 *result* field in the response

The *subfunction_1* field specifies the *editing_state*, meaning what to do with the editing changes, as shown below:

editing_state	meaning
70_{16}	Accept the editing changes
60_{16}	Reject the editing changes
all others	reserved for future specification

Table 10-22 *editing_state*

There is no STATUS or NOTIFY ctype for the ACCEPT/REJECT EDITING CHANGES command.

To monitor the status of the disc subunit, the controller can monitor the disc subunit status descriptor.

To be notified of changes in the state of the subunit status, the controller can use the DISC STATUS command.

10.4 ASSOCIATE LIST WITH PLUG

The ASSOCIATE LIST WITH PLUG command is used to establish the relationship between a subunit source, destination plug or plug group and a specified list, for recording or playback operations. The control command has the following format:

	msb							lsb	
opcode	ASSOCIATE LIST WITH PLUG ($D3_{16}$)								*common command header part*
operand[0]	result								
operand[1]	subfunction_1								
operand[2]	reserved								
operand[3]	associate_state_specification								
:									
:									

Table 10-23 ASSOCIATE LIST WITH PLUG command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00_{16}	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF_{16}	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-24 *result* field in the response

The *subfunction_1* field specifies the *association_state*, meaning what association is to be established:

association_state	meaning
00_{16}	set the default list association for the specified plug
01_{16}	set a specified list/plug association
all others	reserved for future specification

Table 10-25 *association_state*

The *association_state_specification* specifies the details of the association to be established. Its format is dictated by the value of *association_state*. The following figures illustrate each *association_state_specification*:

association_state = 00_{16} (set default association)									
operand offset	msb							lsb	
00_{16}	associated_plug								*plug_ identifier_part*
01_{16}									

Table 10-26 association_state = 00_{16} (set default association)

The *associated_plug* is a *plug_identifier_part*, as described above. This plug will be associated with its default list (determined by the subunit).

association_state = 01_{16} (set specified list)									
operand offset	msb							lsb	
00_{16}	associated_plug								*plug_ identifier_part*
01_{16}									
02_{16}	associated_list_id								*descriptor_ identifier_ part*
:									
:									

Table 10-27 association_state = 01_{16} (set specified list)

The *associated_plug* is a *plug_identifier_part*, as described above.

The *associated_list_id* is a *descriptor_identifier_part* as described above.

The specified plug and list will be associated for subsequent disc operations.

For synchro plug group, the *associated_list_id* shall be the ID of a synchronized performance list (synchro list).

10.5 AUTO UPDATE ON/OFF

The AUTO UPDATE ON/OFF command is used to control automatic updates between the temporary contents lists and the disc media. For details, see section 9.5.

The control command has the following format:

	msb							lsb	
opcode	AUTO UPDATE ($D4_{16}$)								*common command header part*
operand[0]	result								
operand[1]	subfunction_1								
operand[2]	reserved								

Table 10-28 AUTO UPDATE ON/OFF command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00_{16}	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF_{16}	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-29 *result* field in the response

The *subfunction_1* field specifies the *update_state*, as shown below:

update_state	meaning
70_{16}	Automatic updating is on
60_{16}	Automatic updating is off
all others	reserved for future specification

Table 10-30 update_state

There is no STATUS or NOTIFY command variation; the state of the auto update feature can be found in the disc subunit status descriptor, and change notifications on the descriptor can indicate if auto update is turned on or off.

10.6 COMBINE

The COMBINE control command is used to concatenate two tracks into a single track. The control command has the following frame:

	msb							lsb	
opcode	COMBINE (41_{16})								*common command header part*
operand[0]	result								
operand[1]	reserved								
operand[2]	reserved								
operand[3]	anchor_object								*descriptor_ identifier_ part*
:									
:									
:	relocated_object								*descriptor_ identifier_ part*
:									
:									

Table 10-31 COMBINE control command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00_{16}	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF_{16}	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-32 *result* field in the response

The *anchor_object* field is a *descriptor_identifier_part* structure. It specifies the “anchor” object for this operation. The anchor object position does not change its object position number (logical position) when this command is executed.

The *relocated_object* field is a *descriptor_identifier_part* structure. It specifies the object that is to be concatenated to the end of the anchor object. Because it is appended to the anchor object, the relocated object loses its logical identity (object position number).

The results of this command depends on the relative positions of the *anchor_object* and the *relocated_object*, as shown in the examples below.

Example: COMBINE (n + 2, n) where anchor_object > relocated_object

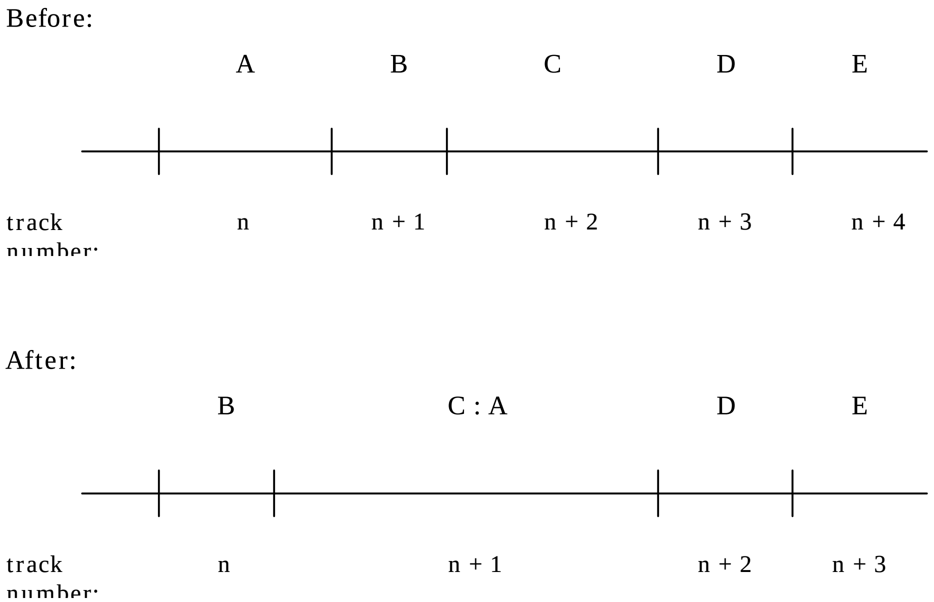


Figure 10-1 combine (where anchor_object > relocated_object)

The above example shows the case where the anchor object position is logically greater than the relocated object position. In this case, there is an effect on object position numbers both before and after the object positions being concatenated. In the example, the contents of B are now reassigned to object position n (formerly contents of A), and the contents of D are now assigned to object position n+2 (formerly contents of C).

Example: COMBINE (n, n+2) where anchor_object < relocated_object

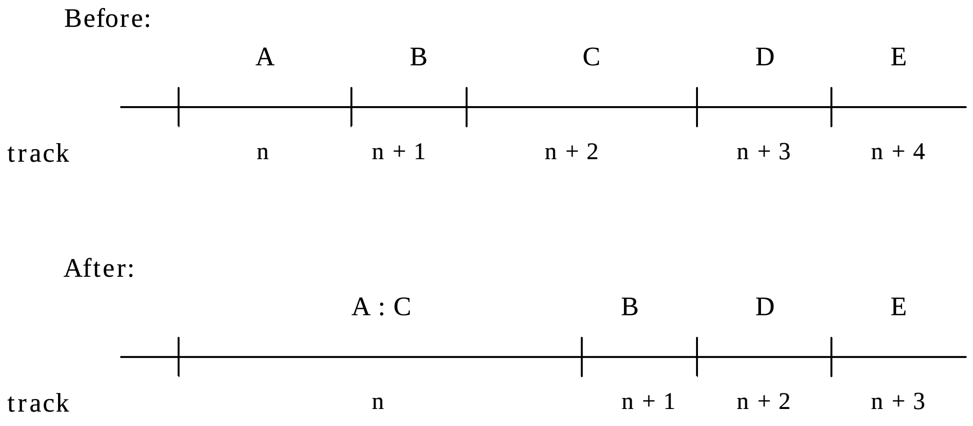


Figure 10-2 combine (where anchor_object < relocated_object)

This example shows the case where *anchor_object* position is less than *relocated_object* position. In this situation, only those object positions following the *anchor_object* are affected.

NOTE: When *anchor_object* and r*elocated_object* refer to the same content item, the target shall REJECT the command.

10.7 CONFIGURE

The CONFIGURE command is used to prepare the subunit for a recording or playback operation. There are several parameters to be configured, depending on the intended action. The control command has the following format:

	msb							lsb	
opcode	CONFIGURE (D1$_{16}$)								*common* *command* *header* *part*
operand[0]	result								
operand[1]	subfunction_1								
operand[2]	reserved								
operand[3]	config_state_specification								
:									
:									

Table 10-33 CONFIGURE command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00$_{16}$	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF$_{16}$	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-34 *result* field in the response

The *subfunction_1* field specifies the *configuration_state*:

configuration_state	meaning
00$_{16}$	reset to the default configuration
01$_{16}$	set a specified configuration
02$_{16}$	change the plug content type configuration
all others	reserved for future specification

Table 10-35 configuration_state

The *config_state_specification* specifies the details of the configuration to be established. Its format is dictated by the value of *configuration_state*. The following figures illustrate each *config_state_specification*:

configuration_state = 00_{16} (reset to the default configuration)									
	msb							lsb	
operand[3]	config_plug								*plug_*
operand[4]									*identifier_part*
operand[5]									
operand[6]	info_block_type								*original*

Table 10-36 configuration_state = 00_{16} (reset to the default configuration)

The *config_plug* is a *plug_identifier_part*, as described above. The controller uses this parameter to specify a subunit source, destination or synchro plug group to be configured. Each plug on the subunit may be configured independently from all others.

The *info_block_type* field specifies the type of info block in the status descriptor for the specified plug, to be reset. The *non_info_block_fields* will be restored to their implementation-defined default values. The only info block types that are currently defined for use in this context are *plug_configuration_info_block* and *playback_order_configuration_info_block*.

configuration_state = 01_{16} (set a specified configuration)									
	msb							lsb	
operand[3]	config_plug								*plug_*
operand[4]									*identifier_part*
operand[5]									
operand[6]	info_block_type								*original*
operand[7]	configuration_information								*orginal*
:									
:									

Table 10-37 configuration_state = 01_{16} (set a specified configuration)

The *config_plug* is a *plug_identifier_part*, as described above. The controller uses this parameter to specify a subunit source, destination or synchro plug group to be configured. Each plug on the subunit may be configured independently from all others.

The *info_block_type* field specifies the type of info block to use for the specified configuration. The only info block types that are currently defined for use in this context are *plug_configuration_info_block* and *playback_order_configuration_info_block*.

The *configuration_information* parameter contains the actual data for the configuration; this is equivalent to the *non_info_block* fields of the specified *info_block_type*.

configuration_state = 02_{16} (change the plug content type configuration)									
	msb							lsb	
operand[3]	config_plug								*plug_*
operand[4]									*identifier_part*
operand[5]	info_block_type								*original*
operand[6]									
operand[7]	AV_object_type								*orginal*

Table 10-38 configuration_state = 02_{16} (change the plug content type configuration)

The *config_plug* is a *plug_identifier_part*, as described above. The controller uses this parameter to specify a subunit source, destination or synchro plug group to be configured. Each plug on the subunit may be configured independently from all others.

The *info_block_type* field specifies the type of info block to use for the specified configuration.

The *AV_object_type* field specifies the type of AV content object the plug is being configured for.

There is no STATUS ctype for the CONFIGURE command. If a controller wants to determine the status of the subunit with respect to its configuration, it can examine the disc subunit status descriptor. For details, please refer to the section titled DISC Subunit Status Descriptor which begins on page 17.

There is no NOTIFY ctype for the CONFIGURE command. If a controller wants to be notified of changes to the state of the subunit's configuration, it can use the DISC STATUS notification command. For details, refer to the description of this command, which begins on page 99.

10.8 CREATE DESCRIPTOR

When creating a child directory object by CREATE DESCRIPTOR, the value of 01_{16} shall be used in subfunction_1: the value of 00_{16} shall not be used.

10.9 DISC STATUS

The DISC STATUS notify command is used to request notification when the status of the disc subunit changes. This change is reflected in the Disc Subunit Status Descriptor structure, as defined on page 17. Note that this command is only defined for the NOTIFY ctype. The notify command has the following format:

	msb							lsb	
opcode	DISC STATUS ($D0_{16}$)								*common*
operand[0]	result								*command*
operand[1]	subfunction_1								*header*
operand[2]	reserved								*part*
operand[3]	status_type_specific								*opcode_*
:									*specific_*
:									*operands*

Table 10-39 DISC STATUS notify command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00_{16}	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF_{16}	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-40 *result* field in the response

The *subfunction_1* field specifies the *status_area*, the “area” of the disc subunit status descriptor for which the controller is requesting notification. The status reporting model defines areas of the status descriptor structure, and each of these areas are treated independently for notification purposes. Thus, if a controller only cares about the general subunit status, or about a specific source or destination plug, it can narrow the scope of the notification messages it receives. This minimizes the amount of time it would take to determine what change has occurred. If a controller cares about changes in several different areas, it can issue several DISC STATUS commands, each specifying an area of interest.

The *status_area* can have one of the following values:

status_area definitions		
Status Area	Value	Meaning
full_status	00_{16}	The entire status descriptor structure is to be monitored for changes.
specified_info_block	01_{16}	An info block in the status descriptor is to be monitored. Any change to that info block, or any of its nested info blocks, results in a change notification.
specified_info_block+mask	02_{16}	A specified info block is to be monitored, but parts of the info block may be “masked out” and changes ignored in this area.
reserved	all others	Reserved for future specification.

Table 10-41 status_area definitions

The *status_type_specific* field contains detailed information specifying the area to be monitored for changes. The format of this field depends on the *status_area* field.

For the *full_status* value, there is no *status_type_specific*; the entire disc subunit status descriptor is to be monitored.

For the *specified_info_block* value, the *status_type_specific* takes the form of an *info_block_reference_path*, as described in reference[1]. Note that the top level of the path is the disc subunit status descriptor (descriptor type specified in section 5). Generally, an info block at any level can be monitored, so it is possible for a controller to be notified of changes in a very narrow scope. However, subunit implementations and media type specifications might put restrictions on the level at which change notification can be supported.

For the *specified_info_block+mask* value, the *status_type_specific* appears as follows:

status_type_specific for status_area = specified_info_block+mask								
	msb						lsb	
operand[3]	info_block_reference_path							*opcode_specific_operands*
:								
:								
:	mask_specifier							*opcode_specific_operands*
:								
:								

Table 10-42 status_type_specific for status_area = specified_info_block+mask

The *info_block_reference_path* is the same as described above.

The *mask_specifier* field indicates which non_info_block fields of the info block will be masked (changes will be ignored). It has the following format:

mask_specifier format								
	msb						lsb	
operand[n]	length_of_mask							
:								
:	1st	2nd	3rd	…	…	…	…	…
:	bit_mask							
:	…	…	…	…	…	…	n-1	n

Table 10-43 mask_specifier format

The *length_of_mask* field specifies the number of bytes in the *bit_mask* field.

The *bit_mask* field contains a number of bytes, where each bit represents the mask for a byte in the non_info_block fields. A byte is masked if its mask bit = 1 (e.g. “mask = true”).

Example: If the 1st bit = 1, then the first byte of the non_info_block fields is masked, and changes to that byte are ignored for status notification purposes.

The remaining bits at the end of the mask that do not correspond to actual bytes in the *non_info_block* fields are ignored.

Note that the disc subunit model places restrictions on the frequency of change notifications, in order to optimize overall system network performance. For details, please refer to section 7.1.4 Updating the Status Information on page 18.

10.10 DIVIDE

The DIVIDE control command is used to split a track into two separate tracks at a specified location. The control command has the following frame:

	msb						lsb	
opcode	DIVIDE (42_{16})							*common*
operand[0]	result							*command*
operand[1]	reserved							*header*
operand[2]	reserved							*part*
operand[3]	divide_location							*edit_location_*
:								*indicator_*
:								*part*

Table 10-44 DIVIDE control command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00_{16}	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF_{16}	unknown	an unknown error occurred
	all other values		reserved for future specification

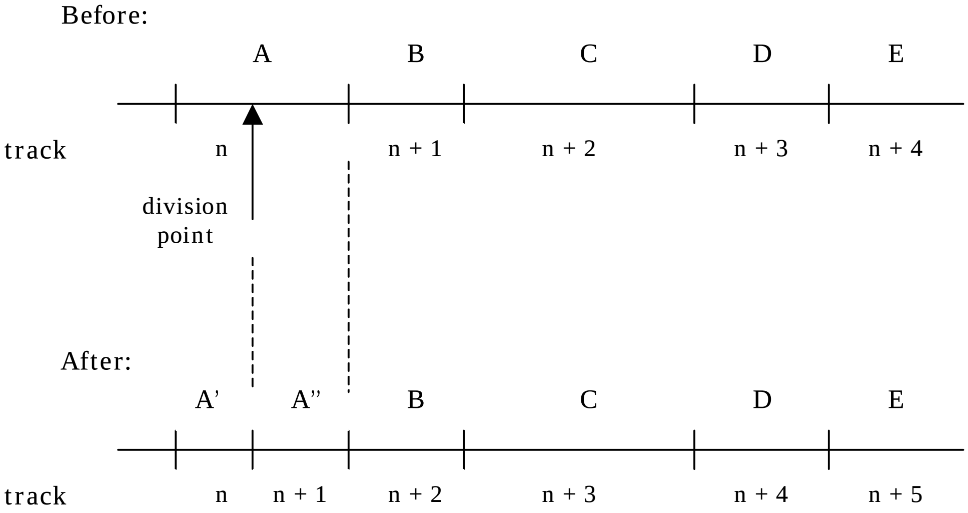
Table 10-45 *result* field in the response

The *divide_location* field is an *edit_location_indicator_part* structure. It specifies the point where the division is to be made.

The following diagram illustrates the DIVIDE operation:

Figure 10-3 DIVIDE

Example: DIVIDE (n)



Note that the contents of the original object position n were divided, but the contents of all other object positions were unaffected. Note also that the object position numbers assigned to all contents following the divided track were incremented as a result of the operation.

There is no STATUS or NOTIFY variant for the DIVIDE command. Controllers can examine the contents list hierarchy if desired.

10.11 ERASE

The ERASE control command is used to remove all or a portion of recorded content from the disc. The control command has the following frame:

	msb							lsb	
opcode	ERASE (40_{16})								*common command header part*
operand[0]	result								
operand[1]	reserved								
operand[2]	reserved								
operand[3]	erase_type_specific								*descriptor_ identifier_part / edit_range_ spec_part*
:									
:									

Table 10-46 ERASE control command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00$_{16}$	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF$_{16}$	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-47 *result* field in the response

The *subfunction_1* field specifies the *erase_type*, meaning which variation of the ERASE command is to be performed. The currently defined values for this field are as follows:

erase_type	value	action
complete	00$_{16}$	Erase all content from the disc
specific_object	01$_{16}$	Erase the specified object from the disc
specific_portion	02$_{16}$	Erase the specified range of content from the disc
-------------	all others	reserved for future specification

Table 10-48 erase_type

The *erase_type_specific* field can vary in format, depending on the value of *erase_type*. It can be either a *descriptor_identifier_part* indicating a specific object to erase, or an *edit_range_specification_part* indicating a portion of the media to be erased. When *erase_type* = complete, there is no *erase_type_specific* field in the command frame.

There is no STATUS ctype for the ERASE command. If a controller wants to determine the status of the subunit with respect to the contents, it can examine the disc subunit contents and/or temporary contents list hierarchies.

There is no NOTIFY ctype for the ERASE command. If a controller wants to be notified of changes to the state of the subunit's contents, it can monitor the contents list hierarchy.

10.12 IMPORT/EXPORT MEDIUM

The IMPORT/EXPORT MEDIUM control command is used to bring a disc into or remove a disc from the drive mechanism. This command is only valid for applications using removable media. The control command has the following format:

	msb							lsb	
opcode	IMPORT/EXPORT MEDIUM (C1$_{16}$)								*common command header part*
operand[0]	result								
operand[1]	subfunction_1								
operand[2]	reserved								

Table 10-49 IMPORT/EXPORT MEDIUM control command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00_{16}	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF_{16}	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-50 *result* field in the response

The *subfunction_1* field specifies the *medium_state*:

medium_state	meaning
60_{16}	export
70_{16}	import
all others	reserved for future specification

Table 10-51 medium_state

There is no NOTIFY ctype for the IMPORT MEDIUM and EXPORT MEDIUM commands. If a controller wants to be notified of changes to the state of the subunit, it can use the DISC STATUS notification command. For details, please refer to the description of that command which begins on page 99.

10.13 INCREMENT OBJECT POSITION NUMBER

The INCREMENT OBJECT POSITION NUMBER command is used to increment the object position number during a recording operation. While the subunit is recording, if it receives this command, it creates a new AV content object and a new descriptor for it. This new object is added to the end of the current list being used for recording.

The format of the command frame is as follows:

	msb							lsb	
opcode	INCREMENT OBJECT POSITION NUMBER (51_{16})								*common*
operand[0]	result								*command*
operand[1]	subfunction_1								*header*
operand[2]	reserved								*part*
operand[3]	destination_plug								*plug_*
operand[4]									*identifier_part*

Table 10-52 INCREMENT OBJECT POSITION NUMBER command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00$_{16}$	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF$_{16}$	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-53 *result* field in the response

The *destination_plug* field is a *plug_identifier_part* structure. It specifies which subunit destination plug, and thus which stream, is to be used for the new track content.

10.14 MONITOR

The MONITOR command is used to monitor the contents coming into a destination plug (e.g. listen to what is being recorded). The control command has the following format:

	msb							lsb	
opcode	MONITOR (C6$_{16}$)								*common command header part*
operand[0]	result								
operand[1]	subfunction_1								
operand[2]	reserved								
operand[3]	source_plug								*plug_ identifier_part*
operand[4]									
operand[5]	monitoring_destination_plug								*plug_ identifier_part*
operand[6]									

Table 10-54 MONITOR command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00$_{16}$	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF$_{16}$	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-55 *result* field in the response

The *subfunction_1* field specifies the *monitor_state*:

monitor_state	meaning
60_{16}	Monitoring off
70_{16}	Monitoring on
all others	reserved for future specification

Table 10-56 monitor_state

The *source_plug* field is a *plug_identifier_part* structure. It specifies which subunit source plug will be used to listen to the monitored signal.

The *monitoring_destination_plug* field is also a plug_identifer_part structure. It specifies the destination plug to be monitored.

The current state of monitoring can be found in the disc subunit status descriptor.

10.15 MOVE

The MOVE control command is used to move the contents of a track from one logical position (the source) to another (the destination). The control command has the following frame:

	msb							lsb	
opcode	MOVE (43_{16})								*common*
operand[0]	result								*command*
operand[1]	reserved								*header*
operand[2]	reserved								*part*
operand[3]	source_object								*descriptor_*
:									*identifier_*
:									*part*
:	destination_object								*descriptor_*
:									*identifier_*
:									*part*

Table 10-57 MOVE control command

The operands of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00_{16}	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF_{16}	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-58 *result* field in the response

The *source_object* field is a *descriptor_identifier_part* structure that specifies the object, by its position in the source list, which is to be moved.

The *destination_object* is a *descriptor_identifier_part* structure that specifies where the object is to be moved. The *destination_object* should be an existing object position number value on the disc – the source object will be moved to this new position. If it specifies a non-existent number, then the target should place the item at the end of the list.

To place the item at the beginning of the list, specify a value of all 00_{16} bytes for the destination position. To place it at the end of the list, specify a value of all FF_{16} bytes.

There is no STATUS ctype for the MOVE command. If a controller wants to determine the status of the subunit contents, it can examine the contents list hierarchy.

There is no NOTIFY ctype for the MOVE command. If a controller wants to be notified of changes to the state of the subunit's contents, it can monitor the contents list hierarchy.

The following example illustrates the MOVE command:

Example: MOVE (n + 3, n + 1)

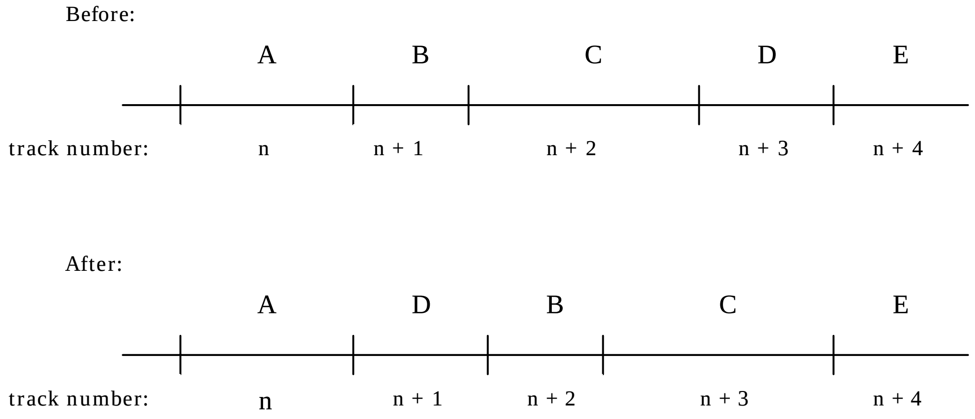


Figure 10-4 MOVE

10.16 OBJECT NUMBER SELECT

The OBJECT NUMBER SELECT command (also called ONS for short) is defined as a common unit and subunit command; each subunit type can define unique subfunctions and usage for the command in addition to the standard definition. For details on the general ONS command, please see reference [1].

The disc subunit uses the ONS command to output an entire AV object(s) with an indication of the actual result. This is different from the normal PLAY command; when PLAY is triggered, an ACCEPTED response is sent immediately, as long as the conditions for playing are satisfied (a disc is inserted, etc.). This kind of operation makes sense for streaming content, because it is reasonable to play only part of the content, such as a few minutes from the middle of a movie.

However, for AV content objects such as digital still images or textual objects, it is critical that the entire object be transmitted in order for the operation to be useful. The ONS command is used for this purpose.

The ONS command can be used to select and transmit several items with one command. The response frame is returned after the last one is successfully transmitted, or at the point where a transmission error occurs.

The following diagram illustrates the basic process of ONS:

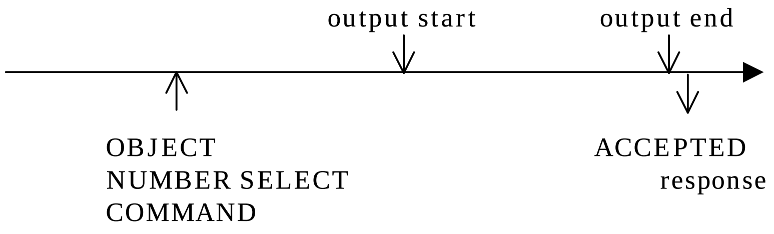


Figure 10-5 basic process of ONS

If a bus reset occurs during transmission, the controller that issued the ONS command should examine the current position of the source plug in the disc subunit status descriptor. This will indicate the progress of the transmission operation.

10.16.1 The Disc Subunit ons_selection_specification Stucture

This section describes the *ons_selection_specification* structure of the OBJECT NUMBER SELECT command. In order to understand this material, it is necessary to understand the basic ONS command, which is described in reference [1].

All of the fields are as described for the general *ons_selection_specification* in the ONS command description

For the disc subunit, the target field of the *ons_selection_specification* shall be specified as a "don't care" path, with no children.

When the *specifier_type_flag* is zero (object is referenced by its list ID and object position), then the target field shall have the following format:

target field ("don't care" specification)	
address offset	contents
00 00_{16}	list_ID
:	
:	
:	object_position
:	
:	
:	number_of_children = 00_{16}

Table 10-59 target field ("don't care" specification)

The number of bytes for the *list_ID* and *object_position* are specified by the *size_of_list_ID* and *size_of_object_position* fields of the subunit identifier descriptor.

When the *specifier_type_flag* is one (object is referenced by its list type and object ID), then the target field shall have the following format:

target field ("don't care" specification)	
address offset	contents
00 00_{16}	list_type
:	target_object_reference (object ID)
:	
:	
:	number_of_children = 00_{16}

Table 10-60 target field ("don't care" specification)

In order to select many items in the same command, the controller can specify the value for *number_of_ons_selection_specifications* in the command frame, and then provide that many specifications (one for each object).

10.16.2 Disc Subunit Behavior

The disc subunit defines the following rules for the response mechanism to the ONS command. Note that some of these may be slightly different than the general ONS command description.

When the disc subunit receives the ONS command, it shall return an INTERIM response (assuming that the command can be carried out). The final response shall be ACCEPTED if all of the objects are transmitted successfully, or REJECTED if there is a failure.

The ACCEPTED response frame shall be as described for the general ONS command.

In the REJECTED response frame, the ons_selection_specification structure for the item that failed shall be returned (if the original command included several items, they shall NOT be returned in the response frame). The status field shall be updated to indicate why the failure occurred, if it can be determined. The following diagram illustrates the REJECTED response frame for the ONS command.

	msb							lsb
opcode	OBJECT NUMBER SELECT ($0D_{16}$)							
operand[0]	source_plug							
operand[1]	subfunction							
operand[2]	status							
operand[3]	number_of_ons_selection_specifications **(n = 1)**							
operand[4]	ons_selection_specification[0] (for the item that failed to transmit successfully)							
:								
:								
:								

Table 10-61 REJECTED response frame for the ONS command

Controllers may use the STATUS command to find out what objects were selected for the plug. The ONS STATUS command works as described for the general ONS command.

Controllers may use the NOTIFY command to find out when the operation has completed. The subunit shall send a CHANGED response only when the operation terminates, either by success or failure. Even though several different items may be transmitted sequentially, the transition from one item to the next does not constitute a “change”, so the subunit shall not send a CHANGED response when it finishes one of the items and begins transmitting the next. The NOTIFY command also works as defined in the general ONS specification.

10.17 PLAY

The PLAY control command is used to begin a playback operation. If the controller has used the SEARCH command to locate a particular location on the media, such as the beginning of a track or an arbitrary location, then playback is started from that point. If not, playback is started from the beginning of the AV content area on the media.

Note: controllers may use the ASSOCIATE LIST WITH PLUG command before beginning the play operation, to specify which list to play from.

The format of the PLAY control command frame is as follows:

	msb							lsb	
opcode	PLAY ($C3_{16}$)								*common command header part*
operand[0]	result								
operand[1]	subfunction_1								
operand[2]	reserved								
operand[3]	source_plug or plug group id								*plug_ identifier_part*
operand[4]									

Table 10-62 PLAY control command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00_{16}	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF_{16}	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-63 *result* field in the response

The *subfunction_1* field specifies the *play_state*, how the command is to be performed. The following values are defined (these are mostly the same as the trick modes for the VCR subunit PLAY command):

Playback Mode	Value	Support level	Description
NEXT FRAME	30_{16}	O	Playback the next sequential frame or field
SLOWEST FORWARD	31_{16}	O	Playback at a special effect speed described in detail below
SLOW FORWARD 6	32_{16}	O	
SLOW FORWARD 5	33_{16}	O	
SLOW FORWARD 4	34_{16}	O	
SLOW FORWARD 3	35_{16}	O	
SLOW FORWARD 2	36_{16}	O	
SLOW FORWARD 1	37_{16}	O	
X1	38_{16}	O	Playback at normal speed
FAST FORWARD 1	39_{16}	O	Playback at a special effect speed described in detail below
FAST FORWARD 2	$3A_{16}$	O	
FAST FORWARD 3	$3B_{16}$	O	
FAST FORWARD 4	$3C_{16}$	O	
FAST FORWARD 5	$3D_{16}$	O	
FAST FORWARD 6	$3E_{16}$	O	
FASTEST FORWARD	$3F_{16}$	O	
PREVIOUS FRAME	40_{16}	O	Playback the previous sequential frame or field
SLOWEST REVERSE	41_{16}	O	Playback in reverse at a special effect speed described in detail below
SLOW REVERSE 6	42_{16}	O	
SLOW REVERSE 5	43_{16}	O	
SLOW REVERSE 4	44_{16}	O	
SLOW REVERSE 3	45_{16}	O	
SLOW REVERSE 2	46_{16}	O	
SLOW REVERSE 1	47_{16}	O	
X1 REVERSE	48_{16}	O	Playback at normal speed in reverse
FAST REVERSE 1	49_{16}	O	Playback in reverse at a special effect speed described in detail below
FAST REVERSE 2	$4A_{16}$	O	
FAST REVERSE 3	$4B_{16}$	O	
FAST REVERSE 4	$4C_{16}$	O	
FAST REVERSE 5	$4D_{16}$	O	
FAST REVERSE 6	$4E_{16}$	O	
FASTEST REVERSE	$4F_{16}$	O	
REVERSE	65_{16}	O	Playback at normal speed in reverse
REVERSE PAUSE	$6D_{16}$	O	Pause in reverse playback
FORWARD	75_{16}	M	Playback at normal speed
FORWARD PAUSE	$7D_{16}$	M	Pause in playback

Table 10-64 playback mode

The disc subunit support level for PLAY, mandatory (M), recomended (R), or optional (O), varies according to both the playback mode requested and the capabilities of the disc subunit. If the subunit does not understand the native format of the data (e.g., the subunit is just a bit recorder), then it is not required to support any of the trick modes. If it does, then it should abide by the support levels indicated in the table.

If there is no disc media in the subunit, then the subunit shall REJECT the command and return the appropriate response frame (which is the same as the command frame).

Speed variations in either a forward or reverse playback direction are collectively referred to as trick play modes. A disc subunit is not required to support any of the trick play modes. However, if trick play modes are supported, then a disc subunit shall implement them as follows:

> There are four groups of trick play modes: slow forward, fast forward, slow reverse and fast reverse. A disc subunit may implement each group inendently.
>
> a) If a disc subunit implements a trick play group, it should implement the basic playback option, i.e., either SLOWEST or FASTEST in the direction implemented. PLAY control commands with a subfunction that specifies a SLOW *n* or FAST *n* playback mode may be rejected by the disc subunit as NOT IMPLEMENTED. Optionally, the disc subunit may accept all of the SLOW *n* or FAST *n* fields and interpret them as SLOWEST or FASTEST.
>
> b) If a disc subunit implements more than one speed within a trick play group, it should recognize all of the SLOW *n* or FAST *n* playback modes as well as the SLOWEST or FASTEST playback mode. A disc subunit is not required to implement all seven possible playback speeds; it is required only to map all possible playback modes within the trick play group to the speeds it does support. The actual speeds encoded by the playback modes should be subject to one of the following restrictions, as appropriate:
>
> SLOWEST <= SLOW 6 <= SLOW 5 <= SLOW 4 <= SLOW 3 <= SLOW 2 <= SLOW 1 <= X1
>
> **OR**
>
> X1 <= FAST 1 <= FAST 2 <= FAST 3 <= FAST 4 <= FAST 5 <= FAST 6 <= FASTEST

The *source_plug* field is a *plug_identifier_part* structure. It specifies which subunit source plug shall carry the output stream.

There is no STATUS ctype for the PLAY command. If a controller wants to determine the status of the subunit with respect to the disc transport actions, it can examine the disc subunit status descriptor. For details, please refer to the section titled DISC Subunit Status Descriptor which begins on page 17.

There is no NOTIFY ctype for the PLAY command. If a controller wants to be notified of changes to the state of the subunit, it can use the DISC STATUS notification command. For details, please refer to the description of this command which begins on page 99.

10.18 RECORD

The RECORD control command is used to record an AV stream onto the disc. The control command has the following frame:

	msb							lsb	
opcode	RECORD ($C2_{16}$)								*common command header part*
operand[0]	result								
operand[1]	subfunction_1								
operand[2]	subfunction_2								
operand[3]	destination_plug								*plug_ identifier_part*
operand[4]									
operand[5]	new_object_position_number								*original*
:									
:									
:	rec_mode_specification								*control_ position_ indicator_part*
:									
:									

Table 10-65 RECORD control command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00_{16}	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF_{16}	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-66 *result* field in the response

The *subfunction_1* field specifies the *rec_state* for this operation. The values defined for this field are as follows:

rec_state	Value	Action
forward	75_{16}	Record are normal speed
forward pause	$7D_{16}$	Go into REC pause mode
time machine	$7E_{16}$	If the subunit is currently in a mode which supports Time Machine recording, then begin recording from buffer memory (see note*)
xxx	all others	Reserved for future specification

Table 10-67 rec_state

Note*: Time Machine recording refers to a feature where the subunit stores the incoming stream in a buffer. The user can monitor the incoming signal and choose when to trigger the recording operation. When triggered, the subunit records from the buffer, which generally has enough memory to hold a few seconds of data. A common use for this feature is recording from a radio broadcast, where the user is waiting for a commercial to end and the normal program to resume. Since the user often has no advance warning, the program resumes before the user can trigger the record operation. Using Time Machine recording, the first few seconds of the program are still in the buffer and are therefore not lost. The subunit implementation decides which mode (REC pause, stop, etc.) can be used to standby for Time Machine recording.

The *subfunction_2* field specifies the *rec_mode*, which variation of the RECORD command is to be executed. The currently defined values for this field are as follows:

rec_mode	Value	Action
new	00_{16}	Begin recording a new track immediately following the last track. If no tracks exist, this should be the first track.
overwrite	01_{16}	Record a new track beginning at the specified position on the medium.
X	all others	reserved for future specification

Table 10-68 rec_mode

The *destination_plug* field is a *plug_identifier_part* structure; it specifies which destination plug is used to record this stream. Depending on the subunit implementation, it is possible that several independent streams may be recorded simultaneously (requiring separate RECORD commands). The controller must prepare the destination plug for recording by issuing the CONFIGURE command before attempting to record.

The *new_object position_number* field shall be set to the value FF FF_{16} when the controller issues the RECORD control command. The subunit shall assign the new object position number value that is being recorded, and return the new object position value in the ACCEPTED response frame. The format of the ACCEPTED response frame is the same as the control command frame, with the new object position number value. If the command is REJECTED, then the response frame is exactly the same as the control command frame.

In a hierarchical storage model, the subunit may allow recording the new object to any specified list. The controller uses the ASSOCIATE LIST WITH PLUG command to specify which list shall receive the new object which is to be recorded. If the subunit does not support recording to any list in the hierarchy, and requires that all newly recorded objects be initially placed in the root contents list, then it shall return NOT IMPLEMENTED when the controller attempts to issue the configuration command specifying a list other than the root contents list. The controller then knows that the new object appears at the root level, and it must move the object to some other location if desired.

The *rec_mode_specification* field is a control_position_indicator_part structure. Its format depends on the value of *rec_mode*. The following rec_mode_specification fields are defined.

10.18.1 rec_mode = new

There is no *rec_mode_specification* field for rec_mode = new. In this case, the command frame ends after the *new_object_position_number* field.

There is no modification of the original track(s) as a result of the Record (new) operation. All allocation for the AV data is from unused space on the disc. The new object descriptor is placed at the end of the list being used for recording.

10.18.2 rec_mode = overwrite

operand offset	msb						lsb	
00_{16}	start_position							*control_ position_ indicator_part*
:								
:								

Table 10-69 *rec_mode* = overwrite

When *rec_mode* = overwrite, the *rec_mode_specification* field becomes a *control_position_indicator_part*, as shown above.

The *start_position* field specifies the object position where recording is to begin; the contents on the media represented by that object position will be overwritten by the newly recorded contents.

The following diagram illustrates this operation:

Example: RECORD (*overwrite* subfunction)

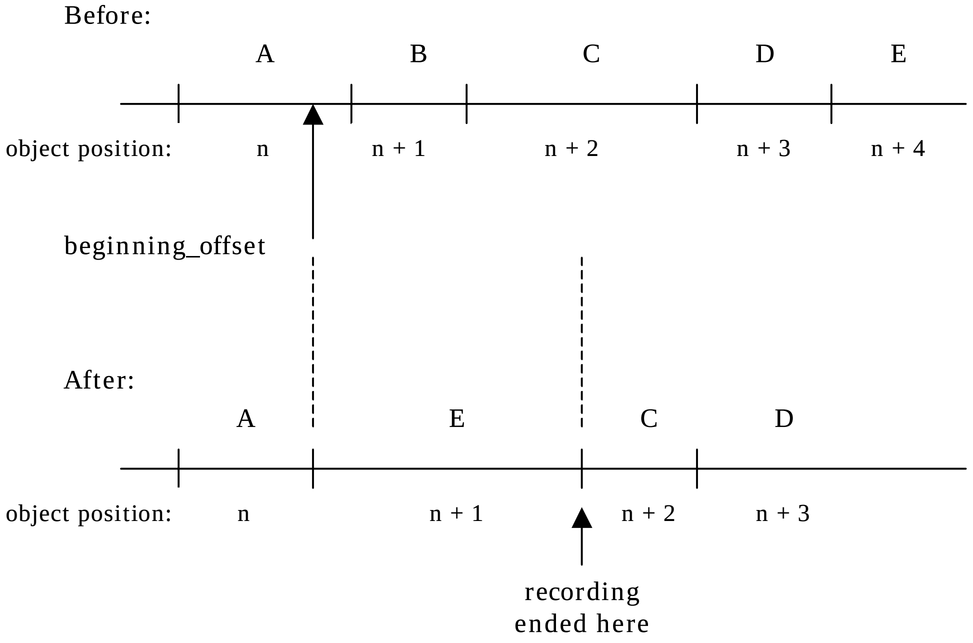


Figure 10-6 record (overwrite subfunction)

Note that this operation caused part of object position n to be truncated, all of object position n+1 was overwritten, and the beginning part of object position n+2 was overwritten. The new contents, represented by E, now occupy object position n+1. The *overwrite* subfunction shall consume used space by overwriting successive tracks, until all used space has been overwritten. If the recording operation continues, then the subfunction may begin consuming free space on the disc. The new object position number shall be assigned as the next sequential object position number following the track in which recording began (see the example diagram).

There is no STATUS ctype for the RECORD command. If a controller wants to determine the status of the subunit with respect to the disc transport actions, it can examine the disc subunit status descriptor. For details, please refer to the section titled DISC Subunit Status Descriptor which begins on page 17.

There is no NOTIFY ctype for the RECORD command. If a controller wants to be notified of changes to the state of the subunit, it can use the DISC STATUS notification command. For details, please refer to the description of this command which begins on page 99.

10.19 RECORD OBJECT

The RECORD OBJECT command is used to record one or more non-streaming AV objects, such as digital still images or textual objects. The motivation for this command is similar to that of the OBJECT NUMBER SELECT command; while streaming contents may be partially recorded, non-streaming objects must be *completely* recorded in order for them to be useful. The RECORD OBJECT is defined for this purpose.

In contrast to the regular RECORD command which returns an ACCEPTED or REJECTED response immediately, the RECORD OBJECT command only returns the final response when the recording operation has finished, either with success or failure.

Note that the rules for the RECORD OBJECT command assumes that it is possible to distinguish the end of the incoming object data.

The following diagram illustrates the process of object recording:

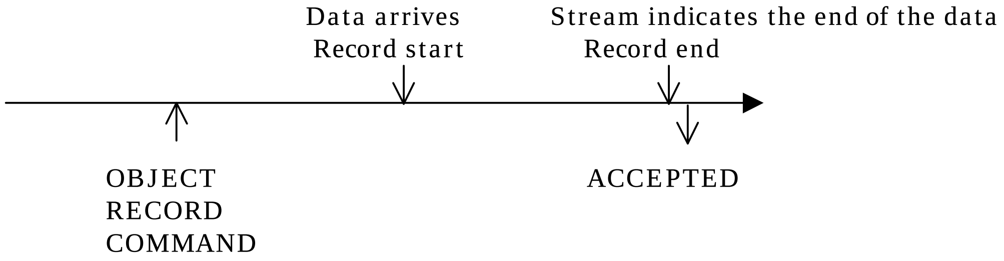


Figure 10-7 process of object recording

Note: [exception] When the value of FF_{16} is specified in the number_of_objects_to_record as subfuntion_1, the subunit shall return an ACCEPTED response at the beginning of the recording same as the regular RECORD command.

If a bus reset occurs during the operation, the controller should examine the current position of the destination plug in the disc status descriptor to check the progress of recording.

The format of the RECORD OBJECT command is as follows:

	msb							lsb	
opcode	RECORD OBJECT (56_{16})								*common*
operand[0]	result								*command*
operand[1]	subfunction_1								*header*
operand[2]	reserved								*part*
operand[3]	destination_plug								*plug_*
operand[4]									*identifier_part*
operand[5]	new_object_position_number								*original*
:									
:									
:	destination_list_id								*descriptor_*
:									*identifier_*
:									*part*

Table 10-70 RECORD OBJECT command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00_{16}	success	Successful recording of all specified objects
	all other values		reserved for future specification
REJECTED	FF_{16}	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-71 *result* field in the response

The *subfunction_1* field specifies the *number_of_objects_to_record*, meaning how many objects will be recorded.

number_of_objects_to_record	meaning
00_{16}	Reserved
01_{16} – FE_{16}	The exact number of objects to be recorded.
FF_{16}	An unspecified number of objects will be recorded.

Table 10-72 number_of_objects_to_record

If the controller does not know how many objects are to be recorded, then it shall specify a value of FF_{16} for this field. In this case, the subunit shall determine when the recording operation is complete (either by running out of storage space, or due to a subunit implementation-specified time out period during which no data is received after the end of an object has been recorded, or some other means).

The *destination_plug* field is a *plug_identifier_part* structure; it specifies which destination plug will be used to record the object(s).

The *destination_list_ID* field is a *descriptor_identifier_part* structure; it specifies which list will receive the newly recorded object(s). The number of bytes for this field is specified by the *size_of_list_ID* field of the subunit identifier descriptor.

The *new_object_position_number* field shall be set to a value of all FF_{16} bytes when the controller issues the RECORD OBJECT control command. The subunit shall assign the first object position value that is being recorded, and return the new value in the ACCEPTED response frame. The format of the ACCEPTED response frame is the same as the control command frame, with the new track number value.

The number of bytes for the *new_object_position_number* field is specified by the *size_of_object_position* field in the subunit identifier descriptor.

The controller may assume that all subsequent objects are stored in subsequent object positions (track values). If the subunit has the ability to record on several destination plugs at the same time, then it shall ensure that the object position assignments are incremental and do not conflict among the recording operations.

One way to do this is to establish new starting track values that are separated by the number of objects to be recorded by each command (e.g. the first command is to record 3 objects, and the subunit starts with position 5; the second command is to record 6 objects, and the starting position is 5 + 3 = 8). However, if a RECORD OBJECT command is received that specifies FF_{16} as the number of objects to be recorded, then the subunit shall not accept any further RECORD OBJECT commands until that operation is complete.

Any controller may use the STATUS command to find out how many objects the subunit is intending to record for a given destination plug, and what starting position has been assigned. The status frame is the same as the CONTROL frame.

The controller that issued the RECORD OBJECT command will receive an INTERIM response initially; when the operation terminates, the subunit shall send the final ACCEPTED or REJECTED response frame.

In the ACCEPTED response frame, the following values shall be returned:

the *result* value to be returned is from the values noted above

the *number_of_objects_to_record* field indicates the number of objects that were actually recorded

the *starting_track_number* is updated with the starting number assigned by the subunit

In the REJECTED response frame, the following values shall be returned:

the *result* field contains an error indicator from the values above

the *number_of_objects_to_record* field is updated with the number of the object at which the failure occurred (or zero if the command was rejected before it began)

the *starting_track_number* contains the initial track number assigned by the subunit, or FF_{16} if the command was rejected before it began

Any controller may use the NOTIFY command to find out when the operation terminates, either with success or failure. The CHANGED response frame shall contain the same information as the final (ACCEPTED or REJECTED) response frame. Note that the controller who issued the RECORD OBJECT command does not need to use the NOTIFY command.

10.20 REHEARSAL

The REHEASAL command is used to play up to three “parts” of the content in a continuous loop. This command is useful for checking the results of an editing operation, such as combine, divide or partial erase. The control command has the following format:

	msb							lsb	
opcode	REHEARSAL ($C7_{16}$)								*common command header part*
operand[0]	result								
operand[1]	subfunction_1								
operand[2]	reserved								
operand[3]	source_plug								*plug_ identifier_part*
operand[4]									
operand[5]	part_1_info								*control_ range_ indicator_part*
:									
:									
:	part_2_info								*control_ range_ indicator_part*
:									
:									
:	part_3_info								*control_ range_ indicator_part*
:									
:									

Table 10-73 REHEASAL command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00_{16}	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF_{16}	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-74 *result* field in the response

The *subfunction_1* field specifies the *rehearsal_state*:

operand offset	msb							lsb
00_{16}	reserved					number_of_parts 00=1 01=2 10=3 11=reserved		repeat = 1

Table 10-75 rehearsal state

The *number_of_parts* field specifies the number of partial contents. Currently up to three parts may be used for this operation.

The *repeat* bit specifies that all the parts are to be performed either one time only, or in a continuous loop.

The *source_plug* field is a plug_identifier_part structure. It specifies which source plug (stream) is to be used for this operation.

The *part_x_info* fields are *control_position_indicator_part* structures. Each one specifies the beginning point and ending point of the partial contents. The *number_of_parts* determines how many of these structures exist in the command frame.

There is no STATUS or NOTIFY ctype for the REHEARSAL command.

To monitor the status of the rehearsal operation, the controller can monitor the disc subunit status descriptor.

To be notified of changes in the state of the subunit status, the controller can use the DISC STATUS command.

10.21 SEARCH

The SEARCH control command is used to find a specified absolute or relative location on the media. If the search command is issued while the source plug is active, the state of the source plug (playing, paused, stopped) after the search location has been found is implementation dependent. However, it is strongly recommended that the state after the search be the same as the state before the search. The control command has the following format:

	msb							lsb	
opcode	SEARCH (50_{16})								*common command header part*
operand[0]	result								
operand[1]	subfunction_1								
operand[2]	reserved								
operand[3]	source_plug								*plug_ identifier_part*
operand[4]									
operand[5]	search_type_specific								*control_ position_ indicator_part / original*
:									
:									

Table 10-76 SEARCH control command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00_{16}	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF_{16}	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-77 *result* field in the response

The *subfunction_1* field specifies the *search_type*. The *search_type* defines the type of search operation, and therefore the format of the *search_type_specific* fields. The following table defines the *search_type* values:

search_type	Meaning
00_{16}	position
01_{16}	absolute_unit
10_{16}	relative_unit
all other values	reserved for future specification

Table 10-78 search_type

The *source_plug* field is a *plug_identifier_part* structure. It specifies which source plug (stream) shall be used.

The format of the *search_type_specific* field can be either a *control_position_indicator_part* structure, or a “non-standard” structure as shown below. The format depends on the value specified for *search_type*:

search_type = 00_{16} (position)									
operand offset	msb							lsb	
00_{16}	indicator_type								*control_ position_ indicator_part*
01_{16}	indicator_type_specific								
:									
:									

Table 10-79 search_type = 00_{16} (position)

When *search_type* = position, the *search_type_specific* field takes the form of a *control_position_indicator_part* structure. For this structure, the following *indicator_type* values can be used:

indicator_type	Meaning
00_{16}	relative HMSF count
01_{16}	relative segment count
02_{16}	absolute HMSF count
03_{16}	relative byte count
04_{16}	absolute byte count
all other values	reserved for future specification

Table 10-80 indicator_type

The indicator_type_specific field bytes are based on the indicator_type. Their formats are defined by the *control_position_indicator_part* specification in section 10.2.2.3.

search_type = 01_{16} (absolute unit)									
operand offset	msb							lsb	
00_{16}	measurement_unit								*original*
01_{16}	count								
:									
:									

Table 10-81 search_type = 01_{16} (absolute unit)

When *search_type* = absolute unit, the *search_type_specific* field takes a non-standard form, as shown.

The *measurement_unit* field specifies the type of "land mark" used for this operation. The following values are defined for this field:

Measurement Unit	Value	Meaning
track	00_{16}	Search to a track boundary as specified in the fields.
segment	01_{16}	A segment boundary
hour	02_{16}	An hour boundary.
minute	03_{16}	A minute boundary.
second	04_{16}	A second boundary.
frame	05_{16}	A frame boundary.
------	all others	Reserved for future specification.

Table 10-82 *measurement_unit*

The *count* field is a multiplier. For example, if the *measurement_unit* specifies a segment, and the *count* specifies 5, then the stream shall be moved to 5th segment boundary. The *count* 0 specifies the beginning of the stream. The number of bytes used for the *count* field is determined by the *size_of_object_position* field in the subunit identifier descriptor.

search_type = 10_{16} (relative unit)									
operand offset	msb							lsb	
00_{16}	measurement_unit								*original*
01_{16}	direction	count							
:									
:									

Table 10-83 search_type = 10_{16} (relative unit)

When *search_type* = relative unit, the *search_type_specific* field takes a non-standard form, as shown.

The *measurement_unit* field is as described above for the absolute unit *search_type*.

The *direction* bit specifies whether the search should be forward (= 1) or backward (= 0).

The *count* field is multiplier for the *measurement_unit*.

For track boundary searches, the destination is always the beginning of a track. A *direction* and *count* of “backward by one” would search to the beginning of the current track. Values specifying “forward by one” would search to the beginning of the next track. A count value of 0 is REJECTED for the relative unit search type.

The number of bytes used for the *count* field is determined by the *size_of_object_position* field in the subunit identifier descriptor.

There is no STATUS ctype for the SEARCH command. If a controller wants to determine the status of the subunit with respect to the disc transport actions, it can examine the disc subunit status descriptor. For details, please refer to the section titled DISC Subunit Status Descriptor which begins on page 17.

There is no NOTIFY ctype for the SEARCH command. If a controller wants to be notified of changes to the state of the subunit, it can use the DISC STATUS notification command. For details, please refer to the description of this command which begins on page 99.

10.22 STOP

The STOP control command is used to stop the flow of data on a specified source or destination plug. The control command has the following format:

	msb							lsb	
opcode	STOP ($C5_{16}$)								*common command header part*
operand[0]	result								
operand[1]	reserved								
operand[2]	reserved								
operand[3]	which_plug								*plug_ identifier_part*
operand[4]									

Table 10-84 STOP control command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00_{16}	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF_{16}	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-85 *result* field in the response

The *which_plug* field is a *plug_identifier_part* structure, indicating which subunit source, destination or synchro plug group to stop.

There is no STATUS ctype for the STOP command. If a controller wants to determine the status of the subunit with respect to the disc transport actions, it can examine the disc subunit status descriptor. For details, please refer to the section titled DISC Subunit Status Descriptor which begins on page 17.

There is no NOTIFY ctype for the STOP command. If a controller wants to be notified of changes to the state of the subunit, it can use the DISC STATUS notification command. For details, please refer to the description of this command which begins on page 99.

10.23 UNDO

The UNDO command is used to get the subunit condition back as before the editing command was executed. If this command is implemented, the minimum support level is to allow undo for the following editing commands: ERASE, DIVIDE, COMBINE, MOVE, WRITE DESCRIPTOR and WRITE INFO BLOCK. Subunit implementations are free to add undo support for any other command if desired.

The number of operations that the subunit can undo is an implementation choice.

The opcode of the command that will be undone is indicated in the general area of the disc subunit status descriptor.

The control command has the following format:

	msb							lsb	
opcode	UNDO (44_{16})								*common command header part*
operand[0]	result								
operand[1]	reserved								
operand[2]	reserved								

Table 10-86 UNDO command

The fields of the *common_header_part* are as described above.

The *result* field in the response may have one of the following values:

response frame type	result	result code name	meaning
ACCEPTED	00_{16}	success	Successful completion
	all other values		reserved for future specification
REJECTED	FF_{16}	unknown	an unknown error occurred
	all other values		reserved for future specification

Table 10-87 *result* field in the response

If there are no supported commands to be undone, then a REJECTED response frame shall be returned.

There is no STATUS or NOTIFY ctype for the UNDO command. Controllers can examine and monitor the disc subunit status descriptor if desired.

11. Disc Subunit Information Block Specifications

This section contains detailed specifications for the disc subunit-specific information block type definitions.

The following table contains the AV/C disc subunit-specific info block types:

Disc Subunit-specific Information Block Types		
info block type	info block name	comments
80 00$_{16}$	artist_info_block	
80 01$_{16}$	genre_info_block	
80 02$_{16}$	disc_capacity_info_block	
80 03$_{16}$	AV_object_type_specific_capacity_info_block	
80 04$_{16}$	AV_content_identifier_info_block	A unique identification value for an AV content object, assigned by the object creator.
80 05$_{16}$	disc_catalog_code_info_block	
80 06$_{16}$	file_format_info_block	Describes the format of a file on storage media.
80 07$_{16}$	audio_recording_parameters_info_block	Information about the recording of an audio object.
80 08$_{16}$	synchro_performance_list_and_plug_pairs_info_block	Defines sets of {list ID, source plug} pairs for synchronized performances.
80 09$_{16}$	descriptor_reference_info_block	Contains a descriptor_identifier that refers to a descriptor structure.
80 0A$_{16}$	text_database_content_attributes_info_block	Describes an entry in the text database.
80 0B$_{16}$	default_play_list_info_block	Indicates which list is the default for Play operations.
80 0C$_{16}$	text_content_type	Indicates the nature of the text in a textual object (lyrics, etc.).
80 0D$_{16}$	output_start_time_info_block	Specifies the time, based on the beginning of the output transmission.
80 0E$_{16}$	presentaion_start_time_info_block	Specifies the recommended time to present the contents object.
80 0F$_{16}$	presentation_end_time_info_block	Specifies the recommended time to stop presenting the content object.
80 10$_{16}$	content_entry_point_info_block	Specifies trim information - where, in the contents object, to start playing.
80 11$_{16}$	content_exit_point_info_block	Specifies trim information - where, in the contents object, to end playing.
80 12$_{16}$ -80 FF$_{16}$	reserved for future definition	

Table 11-1 Disc Subunit-specific Information Block Types

11.1 Artist Info Block (80 00$_{16}$)

The *artist_info_block* specifies information about the recording artist(s) featured on the disc. If this info block is found inside a specific AV object descriptor, then it describes the artist who created that particular content object. If it's found in the contents list list_specific_info, then it describes the artist for the entire disc. There can be several of these info blocks, one for each artist. Alternatively, this info block can represent a group of people.

The format of the *artist_info_block* is as follows:

artist_info_block	
Address Offset	Contents
00 00$_{16}$	compound_length
00 01$_{16}$	
00 02$_{16}$	info_block_type =80 00$_{16}$ (artist_info_block)
00 03$_{16}$	
00 04$_{16}$	primary_fields_length
00 05$_{16}$	
00 06$_{16}$	secondary_fields
:	
:	

Table 11-2 artist_info_block

The *secondary_fields* area contains some number of info blocks which help to describe the artist. This structure is an example of using nested info blocks in order to reuse predefined structures. In this case, the *artist_info_block* may contain at least the following info blocks:

Nested Info Block Types for the artist_info_block		
info block type	info block name	meaning for artist_info_block
00 0B$_{16}$	name_info_block	The name of the artist or group represented by this artist_info_block structure. Possibly many name blocks for different languages.
00 0C$_{16}$	description_info_block	A description of the artist or group. Possibly many for different languages.
00 0D$_{16}$	image_info_block	A picture of the artist or group. Possibly many pictures (info blocks).

Table 11-3 Nested Info Block Types for the artist_info_block

As with the rest of the info block structure model, controllers should not treat the presence of other nested info blocks as an error; they should simply ignore those items which they do not understand. They may choose to ignore or make use of those items which they do understand but did not expect to find.

11.2 Genre Info Block (80 01$_{16}$)

The *genre_info_block* provides a user-readable textual description of the genre of the contents (Jazz, Classical, etc.) of the entity that contains this info block (contents list or object descriptor). This structure has the same format as the *name_info_block*, except for the value of the *info_block_type* field.

11.3 Disc Capacity Info Block (80 02$_{16}$)

The *disc_capacity_info_block* contains various pieces of information about the capacity of the installed disc. It is formatted as follows:

disc_capacity_info_block	
Address Offset	Contents
00 00$_{16}$	compound_length
00 01$_{16}$	
00 02$_{16}$	info_block_type = 80 02$_{16}$ (disc_capacity_info_block)
00 03$_{16}$	
00 04$_{16}$	primary_fields_length
00 05$_{16}$	
00 06$_{16}$	capacity_format_indicator
00 07$_{16}$	disc_total_playback_capacity_length
00 08$_{16}$	
00 09$_{16}$	disc_total_playback_capacity
:	
:	
:	disc_maximum_recording_capacity_length
:	
:	disc_maximum_recording_capacity
:	
:	
:	disc_remaining_recording_capacity_length
:	
:	disc_remaining_recording_capacity
:	
:	

Table 11-4 disc_capacity_info_block

The *capacity_format_indicator* field specifies the format of the *disc_total_playback_capacity*, *disc_maximum_recording_capacity*, and *disc_remaining_recording_capacity* fields. If the disc subunit is able to determine the actual time in hours:minutes:seconds:frames format, then those fields should be reported in that format. If the disc subunit does not have this ability, then it should report those fields as a byte count.

The following values are defined for the *capacity_format_indicator* field:

capacity_format_indicator		
Type	Value	Meaning
time	00_{16}	The fields are reported in hours:minutes:seconds:frames format, with hours in the MSB. Encoded as BCD.
bytes	01_{16}	The fields are reported as a raw byte count.
reserved	all others	Reserved for future specification.

Table 11-5 capacity_format_indicator

The *disc_total_playback_capacity_length* field specifies the number of bytes used to encode the *disc_total_playback_capacity* field; it is two bytes in size. This same rule applies for each of the "*length*" fields in this structure.

The *disc_total_playback_capacity* field specifies the entire playback time (or space) used by all AV content on the disc.

The *disc_maximum_recording_capacity* field specifies the maximum recording time (or space) of the AV content area on the disc.

The *disc_remaining_recording_capacity* field specifies the remaining recording time (or space) of the AV content area on the disc.

These capacity fields do not need to be continuously updated by the subunit while a recording operation is taking place. They only need to be updated when the recording is complete, or when a controller attempts to read the descriptor using the READ INFO BLOCK command. Of course there are other situations where this information may change (such as when a track is deleted from the disc).

11.4 AV Object Type-Specific Capacity Info Block (80 03_{16})

The *AV_object_type_specific_capacity_info_block* contains various pieces of information describing the disc recording capacity for each object type. An example would be the total space available for recording Digital Still Image objects on the disc media (note that this value does not specify the maximum size for an object of the given type).

The subunit might include several *AV_object_type_specific_capacity_info_block* structures, one for each supported AV object type. If the subunit does not have the ability to specify this information, then this information block might not be found by a controller.

The information block is formatted as follows:

AV_object_type_specific_capacity_info_block	
address offset	contents
00 00$_{16}$	compound_length
00 01$_{16}$	
00 02$_{16}$	info_block_type = 80 03$_{16}$ (AV_object_type_capacity_info_block)
00 03$_{16}$	
00 04$_{16}$	primary_fields_length
00 05$_{16}$	
00 06$_{16}$	object_type
00 07$_{16}$	capacity_format_indicator
00 08$_{16}$	object_type_specific_total_playback_capacity_length
00 09$_{16}$	
00 0A$_{16}$	object_type_specific_total_playback_capacity
:	
:	
:	object_type_specific_maximum_recording_capacity_length
:	
:	object_type_specific_maximum_recording_capacity
:	
:	
:	object_type_specific_remaining_recording_capacity_length
:	
:	object_type_specific_remaining_recording_capacity
:	
:	

Table 11-6 AV_object_type_specific_capacity_info_block

The *object_type* field specifies which type of disc subunit AV content object, as specified in 8.1, this information block is for.

All of the subsequent fields have the same format and meaning as defined for the *disc_capacity_info_block* structure defined above.

11.5 AV Content Identifier Info Block (80 04$_{16}$)

The AV_content_identifier_info_block contains a unique identification value for this AV content object. Note that this is NOT a serial number; all instances of this content object have the same content identifier value. An example is the ISRC code assigned to an audio track on commercial CD's. This info block has the following format:

AV_content_identifier_info_block	
Address Offset	Contents
00 00$_{16}$	compound_length
00 01$_{16}$	
00 02$_{16}$	info_block_type = 80 04$_{16}$ (AV_content_identifier_info_block)
00 03$_{16}$	
00 04$_{16}$	primary_fields_length
00 05$_{16}$	
00 06$_{16}$	disc_catalog_code_length
00 07$_{16}$	
00 08$_{16}$	disc_catalog_code
:	
:	

Table 11-7 AV_content_identifier_info_block

11.6 Disc Catalog Code Info Block (80 05$_{16}$)

The *disc_catalog_code_info_block* specifies the catalog code for this disc media. If no such code exists, or the subunit is unable to determine the code, then this information block might not be found by the controller. The info block has the following format:

disc_catalog_code_info_block	
Address Offset	Contents
00 00$_{16}$	compound_length
00 01$_{16}$	
00 02$_{16}$	info_block_type = 80 05$_{16}$ (disc_catalog_code_info_block)
00 03$_{16}$	
00 04$_{16}$	primary_fields_length
00 05$_{16}$	
00 06$_{16}$	disc_catalog_code_length
00 07$_{16}$	
00 08$_{16}$	disc_catalog_code
:	
:	

Table 11-8 disc_catalog_code_info_block

The *disc_catalog_code_length* field specifies the number of bytes used for the *disc_catalog_code* field. If for some reason the info block exists, but the disc subunit is unable to determine the catalog code for the media, then this field shall be set to zero and the *disc_catalog_code* field shall not exist.

The *disc_catalog_code* field contains the catalog code data. The format of this field is specific to the media type. For details, please refer to the appropriate disc subunit media type specification.

11.7 Audio Recording Parameters Info Block (80 07$_{16}$)

The audio_recording_parameters_info_block contains information about the recording of an audio object. The info block has the following format:

audio_recording_parameters_info_block	
Address Offset	Contents
00 00$_{16}$	compound_length
00 01$_{16}$	
00 02$_{16}$	info_block_type = 80 07$_{16}$ (audio_recording_parameters_info_block)
00 03$_{16}$	
00 04$_{16}$	primary_fields_length
00 05$_{16}$	
00 06$_{16}$	audio_recording_sample_rate
00 07$_{16}$	audio_recording_sample_size
00 08$_{16}$	audio_compression_mode
00 09$_{16}$	audio_recording_channel_mode

Table 11-9 audio_recording_parameters_info_block

The *audio_recording_sample_rate* field specifies the sample rate configuration, as defined in the following table:

audio_recording_sample_rate	
Value	Frequency (kHz)
00$_{16}$	32
01$_{16}$	44.1
02$_{16}$	48
03$_{16}$	Reserved for future specification
04$_{16}$	96
all other values	Reserved for future specification

Table 11-10 audio_recording_sample_rate

The *audio_recording_sample_size* field specifies the number of bits used for an audio sample.

The *audio_compression_mode* specifies the type of compression, if any, in effect as the audio signal is recorded. The following table specifies the defined values for this field:

audio_compression_mode	
Value	Compression Mode
00_{16}	no compression
01_{16}	AC-3
02_{16}	Reserved for future specification
03_{16}	Reserved for future specification
04_{16}	MPEG-1 Layer 1 data
05_{16}	MPEG-1 Layer 2 or 3 data, MPEG-2 without extension
06_{16}	MPEG-2 data with extension
07_{16}	Reserved for future specification
08_{16}	MPEG-2 Layer 1 low sample rate
09_{16}	MPEG-2 Layer 2 or 3 low sample rate
$0A_{16}$	reserved for DTS
$0B_{16}$	reserved for DTS
$0C_{16}$	reserved for DTS
$0D_{16}$	reserved for DTS
90_{16}	ATRAC
all other values	Reserved for future specification

Table 11-11 audio_compression_mode

The *audio_recording_channel_mode* field specifies how the audio recording channel is configured. It has the following values:

audio_recording_channel_mode	
Value	Recording Channel Mode
00_{16}	stereo
01_{16}	mono
all other values	Reserved for future specification

Table 11-12 audio_recording_channel_mode

11.8 File Format Info Block (80 06_{16})

The *file_format_info_block* describes the format of a file on the disc media. It has the following structure:

file_format_info_block	
Address Offset	Contents
00 00_{16}	compound_length
00 01_{16}	
00 02_{16}	info_block_type = 80 06_{16} (file_format_info_block)
00 03_{16}	
00 04_{16}	primary_fields_length
00 05_{16}	
00 06_{16}	file_format

Table 11-13 file_format_info_block

The following table specifies the currently defined values for file_format:

file_format	format	comments
00_{16}	plain_text	The file is encoded as a plain text.
80_{16}	MD1	Indicates the MiniDisc MD1 file format. See the MD-audio specification for details.
81_{16}	MD2	Indicates the MiniDisc MD2 file format. See the MD-audio specification for details.
all others	------	Reserved for future definition.

Table 11-14 file_format

11.9 Synchronized Performance List and Plug Pairs Info Block ($80\ 08_{16}$)

The *synchro_performance_list_and_plug_pairs_info_block* describes a set of {performance list ID, subunit source plug} pairs, which are used for the performance. It has the following format:

synchro_performance_list_and_plug_pairs_info_block	
Address Offset	Contents
$00\ 00_{16}$	compound_length
$00\ 01_{16}$	
$00\ 02_{16}$	info_block_type = $80\ 08_{16}$ (synchro_performance_list_and_plug_pairs_info_block)
$00\ 03_{16}$	
$00\ 04_{16}$	primary_fields_length
$00\ 05_{16}$	
$00\ 06_{16}$	number_of_performance_list_plug_pairs (n)
$00\ 07_{16}$	performance_list_ID[0]
:	
:	
:	source_plug[0]
:	:
:	performance_list_ID[n – 1]
:	
:	
:	source_plug[n – 1]

Table 11-15 synchro_performance_list_and_plug_pairs_info_block

The *number_of_performance_list_plug_pairs* field specifies the number of {*performance_list_ID[x]*, *source_plug[x]*} field pairs which follow.

The *performance_list_ID[x]* fields each contain the ID of a performance list. The number of bytes in this field is determined by the *size_of_list_ID* field of the subunit identifier descriptor.

The *source_plug[x]* fields each specify a subunit source plug which will be used to play the content objects specified in the associated performance list. This field is one byte, and its format is specified in the general AV/C specification, in the section which describes unit and subunit plug numbering.

11.10 Text Database Content Attributes Info Block (80 $0A_{16}$)

The *text_database_content_attributes_info_block* describes the contents of the text database (see section 9.8 Text Database Lists for more info on the text database concept). When this info block is encapsulated in a text database object descriptor, it describes the text database content related to **that** descriptor (NOT all of the contents of the text database). It has the following format:

text_database_content_attributes_info_block	
Address Offset	Contents
00 00_{16}	compound_length
00 01_{16}	
00 02_{16}	info_block_type = 80 $0A_{16}$
00 03_{16}	(text_database_content_attributes_info_block)
00 04_{16}	primary_fields_length
00 05_{16}	
00 06_{16}	text_database_object_attributes
00 07_{16}	maximum_number_of_characters
00 08_{16}	

Table 11-16 text_database_content_attributes_info_block

The *text_database_object_attributes* field specifies the attributes of the data referred to by this info block (including the character and language codes - if present, and the text).

text_database_object_attributes		
xxxx xxx1	user_modifiable	1 = the user may modify this text 0 = this text is read-only
xxxx xx1x	stored_on_media	1 = this text is stored on the media 0 = this text is stored in the subunit

Table 11-17 text_database_object_attributes

The *maximum_number_of_characters* field specifies a limitation, if any, on the number of **characters**, **not bytes**, for this text database object. In some subunit implementations, or in some media specifications, there may be limits to fields such as disc and track titles, etc.

If the implementation or media specification does not define a per-text-field limit, then this field shall be set to FF FF_{16}.

11.11 Default Play List Info Block (80 $0B_{16}$)

The *default_play_list_info_block* indicates the list ID to be reproduced by default. When there is no list available to be reproduced by default, this info block shall be empty.
It has the following format:

default_play_list_info_block	
Address Offset	Contents
00 00$_{16}$	compound_length
00 01$_{16}$	
00 02$_{16}$	info_block_type = 80 0B$_{16}$
00 03$_{16}$	(default_play_list_info_block)
00 04$_{16}$	primary_fields_length
00 05$_{16}$	
00 06$_{16}$	default_play_list_ID
:	
:	

Table 11-18 default_play_list_info_block

The *default_play_list_ID* field indicates of the list which should be played by default. When there is no list available to be used as the default, then the *primary_fields_length* shall be set to zero, and the *default_play_list_ID* field shall not exist.

11.12 Text Content Type Info Block (80 0C$_{16}$)

The *text_content_type_info_block* specifies an encoded value that describes the contents of a textual descriptor object. It has the following format:

text_content_type_info_block	
Address Offset	Contents
00 00$_{16}$	compound_length
00 01$_{16}$	
00 02$_{16}$	info_block_type = 80 0C$_{16}$
00 03$_{16}$	(text_content_type_info_block)
00 04$_{16}$	primary_fields_length
00 05$_{16}$	
00 06$_{16}$	text_content_type

Table 11-19 text_content_type_info_block

The *text_content_type* field specifies the coded value that describes the nature of the content:

text_content_type	content type	comments
00$_{16}$	lyrics	The lyrics for an audio track.
01$_{16}$	liner notes	The liner notes for an album.
02$_{16}$	artist information	Describes the artist(s).
03$_{16}$	song information	The information about the song
FF$_{16}$	unspecified	The contents do not correspond to any of the pre-defined values in this table.
all other values	------	Reserved for future definition.

Table 11-20 text_content_type

11.13 output_start_time_info_block (80 0D$_{16}$)

The output_start_time_info_block specify the time, based on the beginning of the output transmission of the performance. This structure has the same format as the position_indicator_info_block, except for the value of the info_block_type field.

11.14 presentaion_start_time_info_block (80 0E$_{16}$)

The presentaion_start_time_info_block specifies the intended time for display device to present the content object to the user. This structure has the same format as the position_indicator_info_block, except for the value of the info_block_type field.

11.15 presentation_end_time_info_block (80 0F$_{16}$)

The presentation_end_time_info_block specifies the recommended time to stop presenting the content object to the user. This structure has the same format as the position_indicator_info_block, except for the value of the info_block_type field.

11.16 content_entry_point_info_block (80 10$_{16}$)

The content_entry_point_info_block specifies trim information - where, in the content object, to start and stop playing. This structure has the same format as the position_indicator_info_block, except for the value of the info_block_type field.

11.17 content_exit_point_info_block (80 11$_{16}$)

The content_entry_point_info_block specifies trim information - where, in the content object, to start and stop playing. This structure has the same format as the position_indicator_info_block, except for the value of the info_block_type field.