

C o m p a c t D i s c I n t e r a c t i v e



F u l l F u n c t i o n a l S p e c i f i c a t i o n

M a y 1 9 9 4 e d i t i o n

**SONY**

**PHILIPS**





C o m p a c t D i s c I n t e r a c t i v e



F u l l F u n c t i o n a l S p e c i f i c a t i o n

Ma y 1 9 9 4 e d i t i o n

**SONY**

**PHILIPS**

# CD-I Full Functional Specification

## Preface

---

### **Copyright**

The Compact Disc Interactive Media Full Functional Specification is published by Philips Consumer Electronics B.V., (Eindhoven, The Netherlands) and has been prepared in close cooperation with Sony Corporation (Tokyo, Japan). All rights are reserved. Reproduction in whole or in part is prohibited without express and prior written permission of Philips Consumer Electronics B.V.

### **Disclaimer**

The information contained herein is believed to be accurate as of the date of publication, however, neither Philips Consumer Electronics B.V., nor Sony Corporation will be liable for any damages, including indirect or consequential, from use of the Compact Disc Interactive Media Full Functional Specification or reliance on the accuracy of this document.

### **Licensing**

The various elements and applications, both hardware and software, mentioned in the Compact Disc Interactive Media Full Functional Specifications, are not necessarily licensed under the Compact Disc License Agreement.

Therefore, it is strongly suggested to verify at all times whether the element or application in question is licensed by Philips or any other relevant third party and on what specific conditions.

### **Classification**

The information contained in this document is marked as confidential and shall be treated as confidential according to the provisions of the Agreement through which the document has been obtained.

### **Notice**

For any further explanation of the contents of this document or in case of any perceived inconsistency or ambiguity of interpretation please consult:

Royal Philips Electronics  
System Standards & Licensing  
Licensing Support  
Building SFF-8  
P.O. Box 80002  
5600 JB Eindhoven  
The Netherlands

Fax.: +31 40 2732113

Internet: <http://www.licensing.philips.com>

# CD-I Full Functional Specification

## Table of Contents

---

### CD-I Full Functional Specification

TOC	Table of Contents	TOC
Chapter I	Introduction	I
Chapter II	CD-I Disc Format	II
Chapter III	Data Retrieval Structure	III
Chapter IV	Audio Data Representation	IV
Chapter V	Video Real Time Data Representation	V
Chapter VI	Program Related Data Representation	VI
Chapter VII	Compact Disc Real Time Operating System	VII
Chapter VIII	Base Case System	VIII
Chapter IX	Full Motion Extension	IX
.....		
Appendix I	Glossary of Terms	A I
Appendix II	CD-I Disc Format/Subheader Values	A II
Appendix V	Video Real Time Data Representation	A V
	V.1 CCIR Level Assignment	
	V.2 Guidelines for 525/625 Image Interchange	
Appendix VII	Compact Disc Real Time Operating System	
	VII.1 CD-RTOS Technical Manual	A VII.1
	VII.2 CD-I Peripherals	A VII.2

## CD-I Full Functional Specification

---

This page is intentionally left blank

# CD-I Full Functional Specification

## Table of Contents

---

Appendix VII	VII.3	Real Time File Handling	A VII.3
.....			
Addendum	Addendum to CD-I Full Functional Specification		
	Note 1	Base Case Performance Figures (Video)	AN 1
	Note 2	Base Case Performance Figures (Audio)	AN 2
	Note 3	List of Known Bugs for CD-RTOS v1.1	AN 3
	Note 4	CD Subcode Graphics for CD-I	AN 4
	Note 5	Multi Disc Applications	AN 5

## CD-I Full Functional Specification

---

This page is intentionally left blank



# CD-I Full Functional Specification

Chapter I

Introduction

Table of Contents

---

## **Chapter I Introduction**

	<b>Page</b>
I.1 Scope	I-1
I.2 Structure of the Specification	I-3
I.3 Conventions	I-5
I.4 Recommended vs. Mandatory	I-7

This page is intentionally left blank

# CD-I Full Functional Specification

Chapter I

Introduction

## List of Illustrations

---

<b>Fig No.</b>	<b>Caption</b>	<b>Page</b>
I.1	Bit ordering for 8 bits (example)	I-5
I.2	Byte ordering for 2 bytes (example)	I-5

This page is intentionally left blank

---

## I.1 Scope

### Chapter I Introduction

#### I.1 Scope

A CD-I<sup>1</sup> System is a real-time<sup>2</sup> system capable of playing CD-I discs. Such a system must be capable of decoding the various types of data (i.e. audio, video, program related data) at the rate at which they are delivered from the CD-I disc. As such the format of a CD-I disc must be specified.

The first goal of this document is to define explicitly the CD-I MEDIA SPECIFICATION, which:

- specifies the physical format of a CD-I disc;
- specifies how various information types---audio data, video data, and program-related data---are coded on a CD-I disc;
- specifies how the various information types are organized on a CD-I disc; and
- gives guidelines about the data integrity and tolerances required to use the CD-I medium.

This specification must comply with the specifications defined in the CD-Digital Audio (CD-DA) Specification<sup>3</sup>. Furthermore, it is based on the principle that:

- the CD-I information carrier is of a fixed bandwidth as defined by the CD-DA Specification;

---

<sup>1</sup> CD-I is the name given to both the **Compact Disc Interactive media** (the disc) and the **Compact Disc Interactive media system** (the hardware) that conform to the specifications laid out in this document.

<sup>2</sup> Real-time system means a system whose data flow through each interface is determined by the disc data flow of 75 sectors/second.

<sup>3</sup> CD-DA Specification: System Description Compact Disc - Digital Audio System ('Red Book'), N.V. Philips and Sony Corporation.

### I.1 Scope

---

- the CD-I information carrier can be substituted for a CD-DA information carrier on any optical medium; and
- CD-DA tracks, defined in the CD-DA Specification, may be used on CD-I discs.

The second goal of this specification is to assure that all CD-I and CD-DA discs can play on all CD-I players and, in particular, in real-time. As such, this specification details the minimum characteristics of a system that may bear the CD-I name. This system is called the Base Case system.

The third goal is to assure the extendability of CD-I systems in an application recognizable manner. In the light of this a configuration status descriptor made up of peripheral status descriptors is available to the application so that it can decide the level of functionality that may be offered to the user.

I.2 Structure of the Specification

---

**I.2 Structure of the Specification**

Chapters II to VI are principally CD-I media specification chapters. Chapters II, IV, and V are structured so as to contain:

- an encoding model;
- a decoding model; and
- the coding on disc

for the physical formats, audio data representations, and visual data representations respectively.

Chapter III gives the specification of the data retrieval structure for files while Chapter VI gives a specification of the program-related data representations. Based on a programming model, Chapter VII specifies the CD-RTOS interfaces. This latter chapter defines the interfaces through which an application can access CD-I resources.

Chapter VIII specifies the Base Case System. In particular, Chapter VIII specifies all the functions that need to be supported by the Base Case System as well as the resources that need to be available.

Each of the aforementioned chapters clearly define which coding formats or modules go beyond the CD-I Base Case System capabilities. When coding formats or modules go beyond Base Case capabilities this is clearly indicated between the relevant text by:

```
*****EXTENSION*****  
.  
.  
.  
text  
.  
.  
.  
*****
```

### I.2 Structure of the Specification

---

The approach to compatibility<sup>4</sup> is defined so that all CD-I discs will be playable on CD-I players.

The appendices (numbered to be consistent with the relevant chapter) are mainly used for clarification.

---

<sup>4</sup> Compatibility to the Base Case is, in general, the responsibility of the application. However, for certain applications where compatibility can be achieved by the use of a formatting structure (e.g. Chapter V: Video) an example compatibility format will be given.



## I.3 Conventions

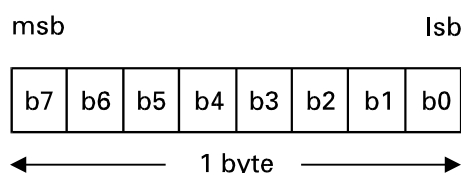
## I.3 Conventions

Unless otherwise indicated in this document the conventions used are as follows:

**Bit ordering**

In this specification, the graphical representation of all multiple-bit quantities is such that the most significant bit is on the left and the least significant bit is on the right.

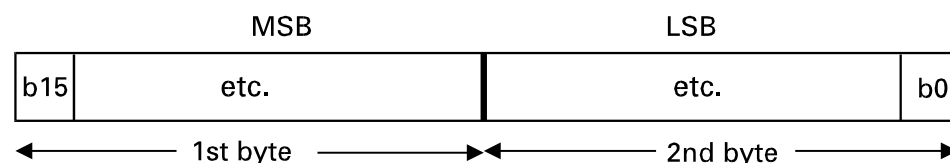
**Figure I.1** Bit ordering for 8 bits (example)

**Byte ordering**

Quantities which require more than 8 bits for their representation are held in more than one byte on disc. For all such quantities, the ordering of bytes on disc (as seen at the interface to the disc driver) is such that the Most Significant Byte is first and the Least Significant Byte is last.

Multiple-byte quantities are represented graphically such that the left-hand-most byte is most significant and the right-hand-most byte is least significant.

**Figure I.2** Byte ordering for 2 bytes (example)



Multiple-byte quantities in memory are represented such that the memory address of successive bytes increases from left to right.

I.3 Conventions

---

**Strings**

Strings are always given between double quotation marks "\_\_\_".

**Hex**

All Hexadecimal values are preceded by a \$.

**Binary**

All binary values are preceded by a %.

**Reserved**

All bits or bytes defined as reserved in this document should be set to zero. Bit fields may be indicated as reserved by the symbol ◆.

I.4 Recommended vs. Mandatory

---

**I.4 Recommended vs. Mandatory**

All specifications given in this document are Mandatory unless specifically defined as **Recommended**.

This page is intentionally left blank

## Table of Contents

**Chapter II CD-I Disc Format**

	<b>Page</b>
II.1 General	II-1
1.1 Scope	II-1
1.2 Conventions	II-2
II.2 CD-I Disc Specification	II-3
2.1 Track Organization	II-3
2.2 Lead-in Area	II-4
2.3 Program Area	II-5
2.3.1 First CD-I track	II-5
2.3.2 Track Layout in the Program Area	II-5
2.3.2.1 Example of only CD-I tracks	II-5
2.3.2.2 Example of CD-I and CD-DA tracks	II-6
2.4 Lead-out Area	II-7
2.5 Subcode P and Q channels	II-8
II.3 CD-I Track Specification	II-14
3.1 CD-I Track Layout	II-14
II.4 CD-I Sector Specification	II-15
4.1 Sector Layout	II-15
4.1.1 General	II-15
4.1.2 Byte Order	II-15
4.2 Scrambling	II-19
4.3 Synchronization Field	II-20
4.4 Header Field	II-21
4.5 Subheader Field	II-22
4.5.1 General Layout	II-22
4.5.2 Subheader Definitions	II-23
4.5.2.1 File Number	II-23
4.5.2.2 Channel Number	II-23
4.5.2.3 Submode	II-23
4.5.2.4 Coding Information	II-23
4.5.3 Submode	II-24

This page is intentionally left blank

Table of Contents

---

	<b>Page</b>
4.6 General Data Classes	II-26
4.7 Form 1 Sector	II-27
4.7.1 General Layout	II-27
4.7.2 Error Detection Code Field	II-27
4.7.3 Error Correction Code Field	II-28
4.8 Form 2 Sector	II-33
4.8.1 General Layout	II-33
4.8.2 Reserved Field	II-33
4.9 Empty and Message Sectors	II-34
4.9.1 Empty Sector	II-34
4.9.2 Message Sector	II-34
II.5 CD-I Encoder Model	II-35
5.1 CD-I Disc Encoder Model	II-35
5.2 CD-I Physical Sector Formatter	II-36
II.6 CD-I Decoder Model	II-37
6.1 CD-I Decoder Model	II-37
6.2 CD-I Sector Processor	II-38
6.3 Byte Order: After the Sector Processor	II-40

This page is intentionally left blank



## List of Illustrations

<b>Fig. no.</b>	<b>Caption</b>	<b>Page</b>
II.1	Example of only CD-I Tracks	II-5
II.2	Example of CD-I and CD-DA Tracks	II-6
II.3	Example of the general structure and coding of the TOC for a CD-I disc with four CD-I tracks and no CD-DA track	II-10
II.4	Example of the encoding of the TOC for disc structure of Fig. II.3	II-11
II.5	Example of the general structure and coding of the TOC for a CD-I disc with one CD-I track and two audio tracks	II-12
II.6	Example of the encoding of the TOC for disc structure of Fig, II.5	II-13
II.7	Relationship between 16 bit samples and data bytes on a CD-I disc a. General b. For CD-I sector	II-16 II-17
II.8	Layout of a CD-I sector	II-18
II.9	Parallel sector synchronized scramble register	II-19
II.10	Layout of the Sync Field of a CD-I sector	II-20
II.11	Layout of the Header Field of a CD-I sector	II-21
II.12	Layout of the Subheader Field of a CD-I sector	II-22
II.13	Bit encoding for the Submode byte	II-24
II.14	Data sector types and classes	II-26
II.15	Layout of a Mode 2 Form 1 sector	II.27
II.16	Bytes contained in P-words (Reed Solomon Column codeword)	II-29
II.17	Parity check matrix $H_p$	II-29

This page is intentionally left blank

List of Illustrations

---

<b>Fig. no.</b>	<b>Caption</b>	<b>Page</b>
II.18	Bytes contained in Q-words (Reed Solomon row codeword)	II-30
II.19	Parity check matrix $H_Q$	II-30
II.20	Interleave map for P and Q-words	II-31
II.21	Rectangular form display of code symbols showing Q-words as rows	II-32
II.22	Layout of a Mode 2 Form 2 sector	II-33
II.23	Example of a Message	II-34
II.24	CD-I Disc Encoder Model	II-35
II.25	CD-I Physical Sector Formatter	II-36
II.26	CD-I Decoder Model	II-37
II.27	CD-I Sector Processor	II-38

This page is intentionally left blank

## **Chapter II CD-I Disc Format**

### **II.1 General**

#### **1.1 Scope**

This chapter gives a specification of the physical format of a CD-I disc. Those parts not specified in this chapter are specified in the CD-DA Specification<sup>1</sup> or in other chapters of the CD-I specification. The CD-I physical format specification is defined in such a fashion that it conforms with the CD-DA Specification.

---

<sup>1</sup> CD-DA Specification = System Description Compact Disc Digital Audio ('Red Book'), N.V. Philips and Sony Corporation

## II.1 General

---

### 1.2 Conventions

This section defines the conventions for the data symbols used in this chapter. A data symbol is an entity made up of  $n$  bits representing a value between 0 and  $2^{n-1}$ . The meaning of indices ( $i, j, k, \dots$ ) is to indicate the sequential order that the data symbols have to be processed in.

Examples of data symbols are:

- $b_i$  - means a binary digit with index  $i$  and a value of 0 or 1.
- $B_i$  - means an 8-bit data byte with index  $i$  and values of 0 to 255.
- $W_i$  - means a 16-bit data word with index  $i$  and values of 0 to 65535.

Bits in a symbol are numbered from 0 upwards. Bit nr. 0 is the least significant bit. The value of a symbol is equal to

$$\sum_{i=0}^{m-1} (b_i * 2^i) ; m = \text{nr. of bits in the symbol.}$$

To specify the relation of symbols an equation style of notation is used here. The values of the left side and the right side have to be equal.

## II.2 CD-I Disc Specification

---

### II.2 CD-I Disc Specification

#### 2.1 Track Organization

The recorded area of the disc is divided into three parts: the lead-in area, the program area and the lead-out area.

The **program area** is divided into a maximum of 99 information tracks, each having a minimum length of 4 seconds, not including the pause length preceding the track. Each information track can be a data track or an audio track.

A **data track** is a track with information encoded as 8-bit wide symbols (bytes) organized in sectors of 2352 bytes.

An **audio track** is a CD-DA track with information encoded as 16-bit wide 2's-Complement numbers. The audio tracks are specified in the CD-DA specification.

The first track in the program area must be a CD-I track (see II.4). All other tracks may be used for either CD-I tracks or CD-DA tracks. The total number of tracks on a CD-I disc cannot be less than 1 or more than 99.

In some CD-I applications, it may be necessary to use more than one CD-I track. This may be done providing that all of the CD-I tracks are grouped together at the beginning of the program area. **This is mandatory.**

If a CD-I track is preceded by another CD-I track the sector structure and addressing in the sector header is not interrupted between these two tracks.

A CD-I track is not allowed to follow a CD-DA track.

The starting point of a CD-DA track is given, in the lead-in area, by the 'Table of Contents', or TOC in the subcode channel Q (see II.2.5).

It is **recommended** that all CD-I information be placed in one track. This track must be the first track of a CD-I disc.

II.2 CD-I Disc Specification

---

**2.2 Lead-in Area**

It is **recommended** that the lead-in area be encoded as a data track (with track number 00) containing Mode 2, Form 1 or Form 2 sectors.



II.2 CD-I Disc Specification

---

2.3 Program Area

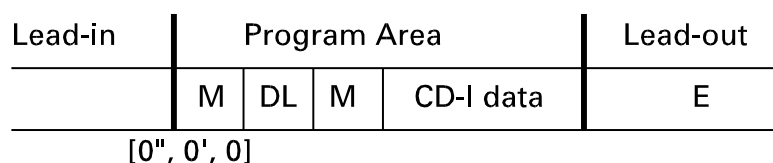
2.3.1 First CD-I track

The first CD-I track (track number 01) on a CD-I disc starts at time 0", 0', 0 ATIME.

2.3.2 Track Layout in the Program Area

2.3.2.1 Example of only CD-I Tracks

Figure II.1 Example of only CD-I Tracks



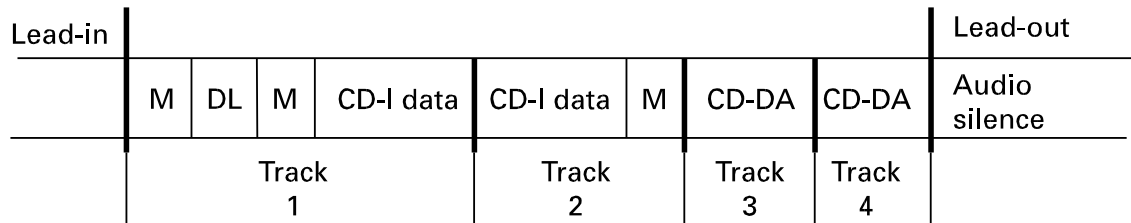
Where DL = Disc Label (from [0", 2', 16] to [0", 2' + S<sub>dl</sub>, 15 + f<sub>dl</sub>])  
 M = Message sectors  
 E = Empty sectors  
 S<sub>dl</sub>, f<sub>dl</sub> = Size disc label in seconds, frames.

In this example the program area contains only one CD-I track covering the whole Program Area. The track starts with 166 message sectors followed by (75 \* S<sub>dl</sub> + f<sub>dl</sub>) sectors for the Disc Label. After the disc label comes a minimum of 2250 message sectors followed by the rest of the CD-I track which contains CD-I data (audio, video, program related data and empty sectors).

## II.2 CD-I Disc Specification

## 2.3.2.2 Example of CD-I and CD-DA Tracks

Figure II.2 Example of CD-I and CD-DA Tracks



In this example the program area contains two CD-I tracks<sup>2</sup> and two CD-DA tracks. The first track is as in II 2.3.2.1. The second and last CD-I track ends with a minimum of 2250 message sectors before the start of the first CD-DA track.

<sup>2</sup> Note that it is **not recommended** to use more than one CD-I track.

II.2 CD-I Disc Specification

---

**2.4 Lead-out Area**

If the last track in the program area of the disc is an audio track, then it is recommended that the lead-out area is encoded as an audio track.

If the last track in the program area of the disc is a CD-I track, then it is recommended that the lead-out area is encoded as a Mode 2 Form 2 (see II.4.8) data track containing empty sectors (see II.4.9.1).

If the lead-out area is encoded as a data track, the data in this lead-out track is sector-structured according to the rules for a data track in Mode 2 Form 2 (see II.4.8).

## II.2 CD-I Disc Specification

---

### 2.5 Subcode P and Q channels

The subcode channels have, in general, the same technical specifications on a CD-I disc as is given in the CD-DA specification with the following exceptions:

- (1) The first CD-I track on the disc is track number one.
- (2) PMIN of POINT \$AO has to be set to a value, which is one higher than the number of CD-I tracks on the disc (2-digit BCD) and which should be less than or equal to 99. CD-I tracks are not indicated in the TOC.
- (3) PSEC of POINT \$AO has to be set to 10 (2 digit BCD coded).  
The PSEC of POINT \$AO is used for the identification of the type of disc.

PSEC of POINT \$AO = %00000000 : means a CD-DA or CD-ROM disc  
 %00010000 : means a CD-I disc  
 All other numbers are reserved.

PFRAME of POINT \$A0 = %00000000

In the control field of POINT \$A0 it is indicated whether the lead-in area is treated as a data track or as an audio track.

- (4) In the lead-in track, channel P = 0. The first CD-I track begins with a start flag of 2 seconds. See the 'Lead-in Track' section of the CD-DA specification (p.40).
- (5) The value of PMIN, PSEC and PFRAME gives the starting point of the audio track number pointed to by POINT. These values give the start position of the track in an absolute time scale (AMIN, ASEC and AFRAME) with an accuracy of +/- one second. The start position of an audio track is the first position with the new track number and X does not equal 00. See the 'POINT' section of the CD-DA specification (p.43).
- (6) The control field of a lead-in Q channel frame is identified as the CONTROL field used in the track that POINT refers to. See pp.41 and 43 of the CD-DA specification.
- (7) PMIN of POINT \$A1 has the value of the last audio track number on the disc if the CD-I disc has CD-DA tracks, otherwise PMIN of POINT \$A1 has a value which is one higher than the number of CD-I tracks on the disc.

### II.2 CD-I Disc Specification

---

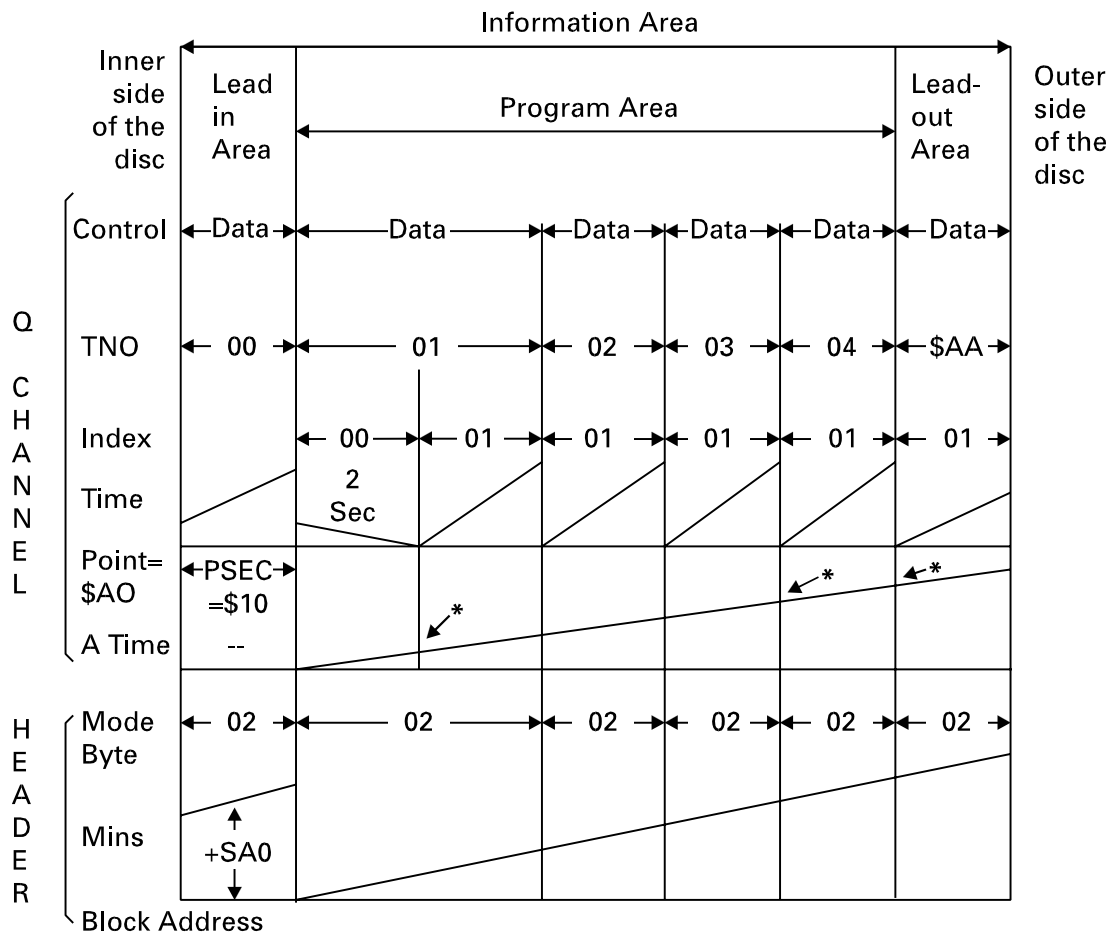
PSEC and PFRAME of POINT \$A1 are zero. The control field of the FRAME with POINT = \$A1 contains "00X0" if the CD-I disc has CD-DA tracks, otherwise the control field contains "01X0" (X = 0 for copy prohibited. X = 1 for copy permitted).

- (8) POINT = \$A2, indicates the starting point of the lead-out track in PMIN, PSEC and PFRAME. In the CONTROL field it is indicated whether the lead-out area is treated as a data track or as an audio track.

Figures II.3 to II.6 give examples of the encoding of the TOC as well as for the general structure of CD-I discs.

II.2 CD-I Disc Specification

Figure II.3 Example of the general structure and coding of the TOC for a CD-I disc with four CD-I tracks and no CD-DA track



\* These points as specified in the Q-channel give the start position of the first and last track or lead-out area in ATime with an accuracy of  $\pm$  one second.

## II.2 CD-I Disc Specification

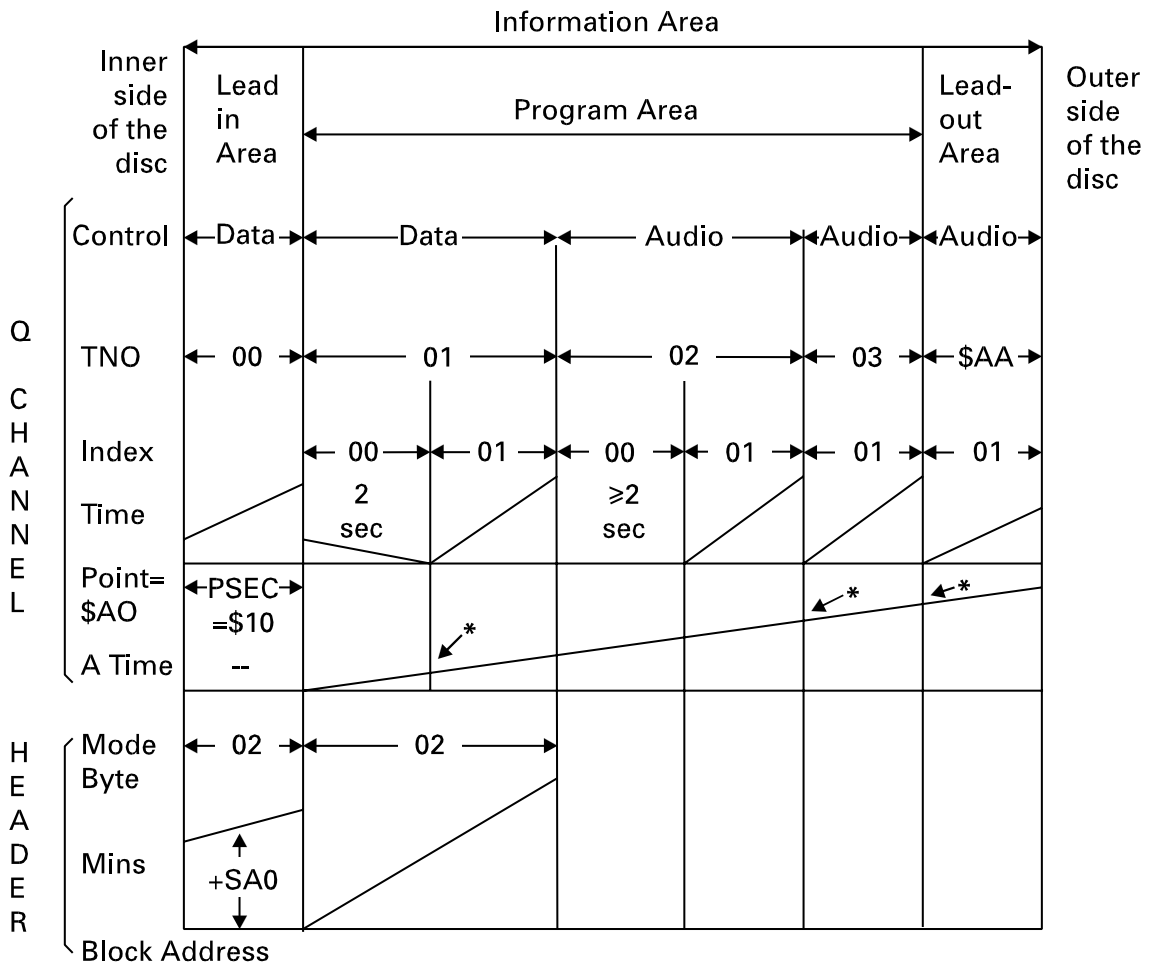
Figure II.4 Example of the encoding of the TOC for disc structure of Fig. II.3

Frame Number	Point	PMIN, PSEC, PFRAME	Control Field
n	\$A0	05, 10, 00	01 x 0
n + 1	\$A0	05, 10, 00	01 x 0
n + 2	\$A0	05, 10, 00	01 x 0
n + 3	\$A1	05, 00, 00	01 x 0
n + 4	\$A1	05, 00, 00	01 x 0
n + 5	\$A1	05, 00, 00	01 x 0
n + 6	\$A2	52, 48, 41	01 x 0
n + 7	\$A2	52, 48, 41	01 x 0
n + 8	\$A2	52, 48, 41	01 x 0
n + 9	\$A0	05, 10, 00	01 x 0
n + 10	\$A0	05, 10, 00	01 x 0

Figures II.3 and II.4 are examples of the encoding of the TOC for a CD-I disc with four CD-I tracks and no CD-DA tracks. The lead-in area and the lead-out area are treated as data tracks.

II.2 CD-I Disc Specification

Figure II.5 Example of the general structure and coding of the TOC for a CD-I disc with one CD-I track (01) and two audio tracks (02, 03) in the program area



\* These points as specified in the Q-channel give the start position of the first and last track or lead-out area in ATime with an accuracy of ± one second.



## II.2 CD-I Disc Specification

Figure II.6 Example of the coding of the TOC for disc structure of Fig. II.5

Frame Number	Point	PMIN, PSEC, PFRAME	Control Field
n	02	10, 15, 12	00 x 0
n + 1	02	10, 15, 12	00 x 0
n + 2	02	10, 15, 12	00 x 0
n + 3	03	16, 28, 63	00 x 0
n + 4	03	16, 28, 63	00 x 0
n + 5	03	16, 28, 63	00 x 0
n + 6	\$A0	02, 10, 00	01 x 0
n + 7	\$A0	02, 10, 00	01 x 0
n + 8	\$A0	02, 10, 00	01 x 0
n + 9	\$A1	03, 00, 00	00 x 0
n + 10	\$A1	03, 00, 00	00 x 0
n + 11	\$A1	03, 00, 00	00 x 0
n + 12	\$A2	52, 48, 41	00 x 0
n + 13	\$A2	52, 48, 41	00 x 0
n + 14	\$A2	52, 48, 41	00 x 0
n + 15	02	10, 15, 12	00 x 0
n + 16	02	10, 15, 12	00 x 0

Figures II.5 and II.6 are examples of the encoding of the TOC for a CD-I disc with one CD-I track (01) and two audio tracks (02, 03) in the Program area. The lead-in area is treated as a data track and the lead-out area is treated as an audio track.

## II.3 CD-I Track Specification

---

### II.3 CD-I Track Specification

#### 3.1 CD-I Track Layout

In a CD-I track the data is divided into CD-I sectors of 2352 sequential bytes each. There are five different types of CD-I sectors depending on the type of information they contain i.e. Audio, Video, Data, Empty and Message Sectors (see II.4.9). Each sector can be uniquely addressed by a BCD-coded absolute time value in the header field of the sector (see II.4.4).

CD-I sectors with time values of (00", 00', 00) up to (00", 02', 15) must be message sectors (see II.4.9.2). Message sectors contain after scrambling, (see II.4.2), the actual CD-DA coded digital audio data.

The sector with a time value of (00", 02', 16) must be a data-sector in Form 1 (see II.4.7).

The sectors from and including time value (00", 02', 16) onward to (00", 02' +  $S_{dl}$ , 15 +  $f_{dl}$ ) are data sectors in Form 1 and are all part of the Disc Label (see III.2) which has a length of  $(75 * S_{dl} + f_{dl})$  sectors.

From the end of the disc label there must be at least 2250 contiguous message sectors (see II.4.9.2). Thereafter, CD-I sectors with CD-I information may start again.

If a CD-I disc contains CD-DA tracks then the first CD-DA track has to be directly preceded by a minimum of 2250 contiguous message sectors.

## II.4 CD-I Sector Specification

---

### II.4 CD-I Sector Specification

#### 4.1 Sector Layout

##### 4.1.1 General

In a CD-I track the data is divided into addressable CD-I sectors of 2352 sequential bytes.

A CD-I sector contains the following data fields:

- Sync field            12 bytes
- Header field         4 bytes
- Subheader field     8 bytes
- Data field            2328 bytes

All data in a CD-I sector, except the synchronization field is scrambled (see II.4.2).

##### 4.1.2 Byte Ordering

The bytes in a CD-I sector are indexed from 0 to 2351. The index also indicates the sequential order in which the bytes have to be processed by the encoder. The layout of a CD-I sector is given in Figures II.7 and II.8.

The bit/word ordering on disc is as defined in the CD-DA specification.

The relationship between CD-I data bytes (B) and CD-DA 16-bit words (W) is:

$$W_{2i} = 256 * B_{4i+1} + B_{4i} \text{ even word}$$

$$W_{2i+1} = 256 * B_{4i+3} + B_{4i+2} \text{ odd word}$$

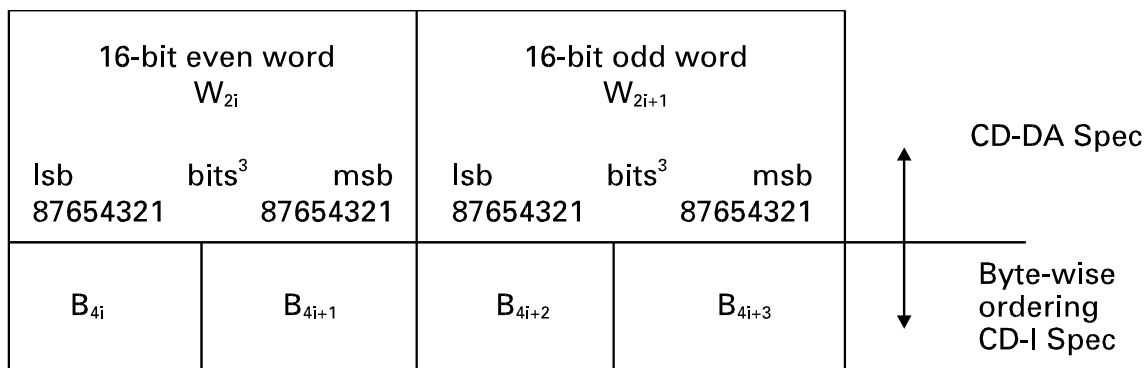
The data rate is such that the index i is incremented nominally 44100 times per second.

The first byte,  $B_0$ , of a CD-I sector is after scrambling the 8 least significant bits of the even indexed 16-bit word (see Figure II.7).

II.4 CD-I Sector Specification

---

Figure II.7a Relationship between 16 bit samples and data bytes on a CD-I disc - General




---

<sup>3</sup> CD-DA Specification bits are ordered from bit 8 (least significant) to bit 1 (most significant).

II.4 CD-I Sector Specification

Figure II 7b **Relationship between 16 bit samples and data bytes on a CD-I disc - For CD-I sector**

The relationship between 16 bit samples and data bytes on a CD disc			
← 16-bit word even word →		← 16-bit word odd word →	
lsb 87654321	bits <sup>3</sup> 87654321	lsb 87654321	msb 87654321
00000000	11111111	11111111	11111111
11111111	11111111	11111111	11111111
11111111	11111111	11111111	00000000
MINUTES	SECONDS	SECTORS	MODE
BYTE 16	BYTE 17	BYTE 18	BYTE 19
.	.	.	.
.	.	.	.
.	.	.	.
BYTE n	BYTE n+1	BYTE n+2	BYTE n+3
.	.	.	.
.	.	.	.
.	.	.	.
BYTE 2348	BYTE 2349	BYTE 2350	BYTE 2351

} SYNC  
 } HEADER

II.4 CD-I Sector Specification

---

Figure II.8 **Layout of a CD-I sector**

Byte Number	Layout
0 1 · · · 11	Sync 12 bytes
12 13 14 15	Header 4 bytes
16 17 · · · · 2350 2351	Data 2336 bytes

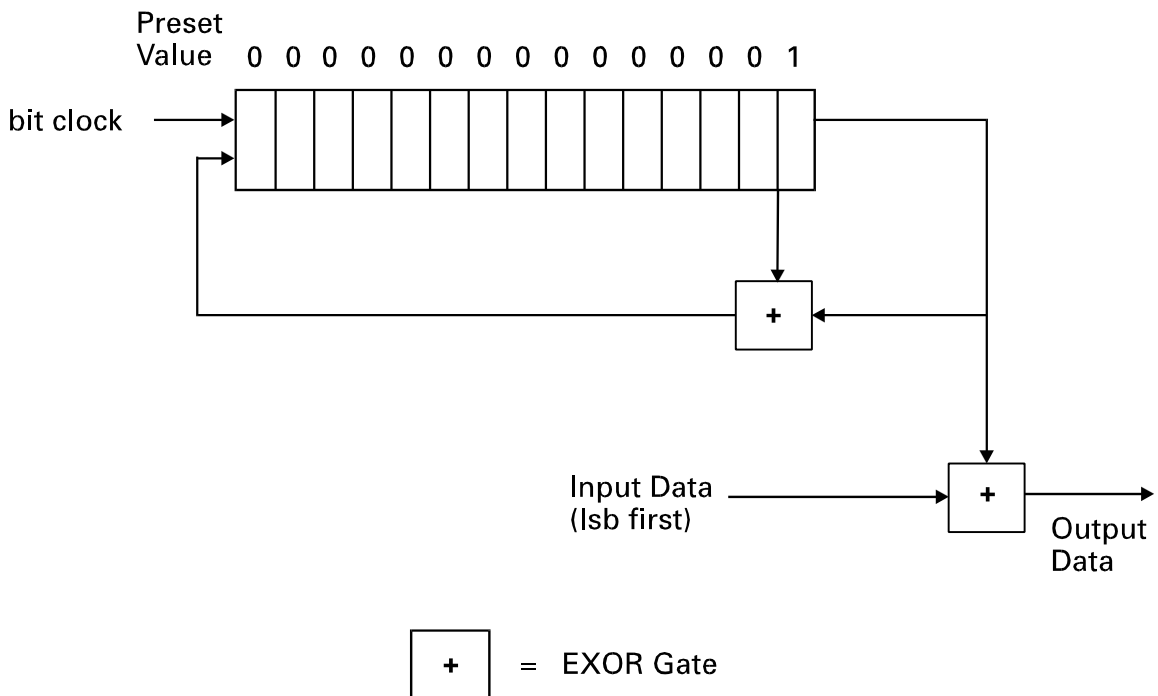
II.4 CD-I Sector Specification

---

4.2 Scrambling

All data in a CD-I sector except the data in the synchronization field is scrambled. The scrambler is of a parallel sector synchronized type. The scramble register is a 15-bit shift register fed back according to the polynomial  $X^{15} + X + 1$ . The contents of this scramble register is EXOR-ed with the serial information bit by bit, taking the least significant bit of the data bytes first. The scramble register is preset with the value %0000 0000 0000 001 after the synchronization field of a data sector. The scramble register is given in Figure II.9.

Figure II.9 Parallel sector synchronized scramble register



II.4 CD-I Sector Specification

---

**4.3 Synchronization Field**

The organization of the synchronization field (12 bytes) is given in Figure II.10 below. All bytes in the synchronization field have the value 255 except the first and last bytes, which have the value 0.

Figure II.10 **Layout of the Sync Field of a CD-I sector**

Layout of the Sync Field	
Byte Number	Byte Value
0	0
1	255
2	255
3	255
4	255
5	255
6	255
7	255
8	255
9	255
10	255
11	0



## II.4 CD-I Sector Specification

---

### 4.4 Header Field

The organization of the header field (4 bytes) is given in Figure II.11 below. The header contains the CD-I sector address (3 bytes) and the mode byte.

The relative time difference of the sector address in the header field to the subcode channel Q ATIME should be less than one second.

Figure II.11 **Bit encoding for the Submode byte**

Layout of the Header Field	
Byte Number	Byte Value
12	Minutes
13	Seconds
14	Sectors
15	Mode

Minutes = a copy of the data contents of AMIN (subcode channel Q)

Seconds = a copy of the data contents of ASEC (subcode channel Q)

Sectors = a copy of the data contents of AFRAME (subcode channel Q)

Mode = 2 must be used for CD-I tracks.

II.4 CD-I Sector Specification

---

**4.5 Subheader Field****4.5.1 General Layout**

The subheader field consists of 8 bytes. The subheader data is double-written for data integrity (see Figure II.12).

The meaning and value of the subheader data is defined in the applicable chapters.

**Figure II.12 Layout of the Subheader Field of a CD-I sector**

Layout of Subheader Field		
Byte Number	Byte Value	Applicable Chapter
16	File number	III
17	Channel number	VII
18	Submode	II
19	Coding information	IV, V, and VI
20	File number	
21	Channel number	
22	Submode	
23	Coding information	

Since the bit allocations of these subheaders are given in various chapters, for clarity and conciseness, Appendix II.1 reiterates these results.

## II.4 CD-I Sector Specification

---

### 4.5.2 Subheader Definitions

#### 4.5.2.1 File Number

The file number is used to identify all sectors that belong to one and the same file. The details of the file number are given in III.4.4 and Appendix II.1.

#### 4.5.2.2 Channel Number

A real-time record may contain several different pieces of information that need be chosen in combination or separately at playback. To facilitate the real-time selection of such information each piece of information may be given a unique channel number. The details of the channel number are given in VII.2.2.3.2 (SS\_Play) and Appendix II.1.2.

#### 4.5.2.3 Submode

The submode byte defines global attributes of a sector as required for the initial selection and allocation of a sector in the system, termination of a file or record, initialization of an additional layer of error correction, and synchronisation. For details see II.4.5.3 and Appendix II.1.3.

#### 4.5.2.4 Coding Information

This byte defines the details of the type of data located in the user area of the sector. As this depends on whether the data is audio, video or program related data its details are given in IV.3.2.4, V.6.3.1 and VI.4.2 and is repeated in Appendix II.1.4.

### 4.5.3 Submode

The submode byte is bit-encoded as shown in Figure II.13

Figure II.13 **Bit encoding for the Submode byte**

Bit Number	Bit Name
7	End Of File (EOF)
6	Real-Time Sector (RT)
5	Form (F)
4	Trigger (T)
3	Data (D)
2	Audio (A)
1	Video (V)
0	End Of Record (EOR)

**End Of File (EOF)** : The last sector of a file is indicated by bit 7 equal to 1. All other sectors have bit 7 equal to zero.

**Real-Time Sector (RT)** : If bit 6 has the value 1 then the data has to be processed without interrupting the real-time behavior of the CD-I system. For example, audio sectors have to be transferred to the ADPCM decoder in real time in order to avoid the overflow or underflow of data.

**Form (F) Sectors** : This bit has a value of 0 for all recorded in Form 1 and a value of 1 for all sectors recorded in Form 2 (see IV, V and VI).

**Trigger (T)** : This bit is used to synchronize the application with various coding information, like visuals to audio, in real time. The bit when set to one generates an interrupt (see VII.4).

**Data (D)** : This bit has a value of 1 for program related data sectors. Otherwise it is zero. When this bit is set to one, the Form bit must be zero.

#### II.4 CD-I Sector Specification

---

Audio (A) : This bit has a value of 1 for audio sectors. Otherwise it is zero. When this bit is set to one the Form bit is also set to one.

Video (V) : This bit has value of 1 for video sectors. Otherwise it is zero.

End Of Record (EOR) : This bit must have the value 1 for the last sector of a logical record. Other-wise it is zero. The use of the EOR bit is only mandatory for real-time records.

Only one of bit 1, bit 2 or bit 3 may have the value 1 at the same time. One of bit 1, bit 2 and bit 3 must be set to the value 1 for all sectors except empty and message sectors. If bits 1, 2 and 3 of the submode byte are zero then the sectors are either empty or message sectors.

**Note:** If a trigger bit and an EOR bit are set in the same sector, and the sector was not selected via the channel mask, the EOR bit will be reset by the CD driver so that it is not acted upon by the driver or the application.

## II.4 CD-I Sector Specification

## 4.6 General Data Classes

Sectors on a CD-I disc may contain two generic classes of data:

- . data with a strong requirement for data integrity where concealment is not possible (e.g. text, computer data)
- . data where errors can be concealed by some interpolation scheme (e.g. audio, video).

The text and computer data, for example, have a strong requirement for an extra layer of error correction beyond the CIRC (cross-interleaved Reed-Solomon code) used on all Compact Discs. For this class of data, error detection and correction should be implemented according to the CD-I EDC/ECC (see II.4.7). The sector structure used for this class of data is called Form 1 and is specified in detail in II.4.7.

The latter class of data has a strong requirement for maximum data throughput per time unit. The sector structure used for this class of data is called Form 2 and is specified in detail in II.4.8.

It should be noted that both Form 1<sup>4</sup> and Form 2 may be used for real-time data. However, for Form 1 real time sectors, priority is given to the real time aspect (and not to data integrity). Therefore only players with built-in hardware support that enables them to correct errors fully transparently in real-time without requiring CPU intervention will correct these sectors. Consequently, a CD-I base case system without real time ECC handling will not correct Form 1 real time sectors.

Allowed combinations of different sector types and classes are given in the table below.

Figure II.14 Data sector types and classes

	Form 1		Form 2	
	RT = 0	RT = 1	RT = 0	RT = 1
Audio	No	No	Yes	Yes
Video	Yes	No	Yes	Yes
Data	Yes	Yes <sup>4</sup>	No	No
Empty	Yes	Yes	Yes	Yes
Message	No	No	Yes	No

<sup>4</sup> Some restrictions apply to using Form 1 for real time data see Appendix II.1.3.

## II.4 CD-I Sector Specification

## 4.7 Form 1 Sector

## 4.7.1 General Layout

A Form 1 sector consists of 2352 bytes. Each byte in the sector can be identified by  $B_i$  where  $i = 0$  to 2351.

The layout of a Form 1 sector is as follows:

Figure II.15 Layout of a Mode 2 Form 1 sector

Byte	Meaning
$B_0$ to $B_{15}$	the synchronization and header fields
$B_{16}$ to $B_{23}$	the subheader field
$B_{24}$ to $B_{2071}$	2048 bytes of data whose content depends on the sector type
$B_{2072}$ to $B_{2075}$	EDC field of 4 bytes
$B_{2076}$ to $B_{2351}$	ECC field

## 4.7.2 Error Detection Code Field

The EDC used is a 32-bit CRC on the data field defined as  $B_{16}$  to  $B_{2071}$ .

The EDC codeword must be divisible by the check polynomial.

The least significant bit of a data byte is used first in the CRC mechanism. The least significant bit ( $X^0$ ) of the EDC parity is located at bit 7 of byte 2075.

The check polynomial is:

$$P(X) = (X^{16} + X^{15} + X^2 + 1) * (X^{16} + X^2 + X + 1)$$

The first factor of the check polynomial is the traditional CRC-16. Depending on the desired level of reliability one can check one or more factors of  $P(X)$ .

## II.4 CD-I Sector Specification

---

### 4.7.3 Error Correction Code Field

The ECC is a kind of product code over GF ( $2^8$ ), bytes being symbols. In this section the bytes are numbered as parts of 16- bit words. The consecutive words in a sector are numbered 0 to 1169, where the numbering starts right after the synchronization pattern. Hence, the whole sector exclusive of the synchronization pattern is protected by the ECC.

The translation rule from bytes into words is:

$$S_N = B_{2N+12} + 256 * B_{2N+13}$$

with  $N = 0$  to 1169 and

$B_{12}$  to  $B_{15}$  must be set to 0

A sector consists of 2 codewords as follows:

- 1 codeword containing the most significant bytes of each word.
- 1 codeword containing the least significant bytes of each word.

This can be envisioned as if one has 2 planes for ECC, 1 plane containing the MSB and the second plane containing the LSB. On each plane the same product code is defined. In each plane one still labels each byte by the word number defined above. The field GF ( $2^8$ ) is generated by the primitive polynomial:

$$P(X) = X^8 + X^4 + X^3 + X^2 + 1$$

The primitive element  $a$  of GF ( $2^8$ ) is :

$$a = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0)$$

msb                      lsb





Figure II.18 **Bytes contained in Q-words (Reed Solomon row codeword)**

$$\begin{array}{l}
 \text{Q-word: } V_Q = \left[ \begin{array}{l}
 S(44 * 0 + 43 * N_Q) \\
 S(44 * 1 + 43 * N_Q) \\
 S(44 * 2 + 43 * N_Q) \\
 S( \quad " \quad ) \\
 S( \quad " \quad ) \\
 S( \quad " \quad ) \\
 S(44 * M_Q + 43 * N_Q) \\
 S( \quad " \quad ) \\
 S( \quad " \quad ) \\
 S( \quad " \quad ) \\
 S(44 * 41 + 43 * N_Q) \\
 S(44 * 42 + 43 * N_Q) \\
 S(43 * 26 + \quad N_Q) \\
 S(44 * 26 + \quad N_Q)
 \end{array} \right]
 \end{array}
 \begin{array}{l}
 N_Q = 0,1,2, \dots, 25 \\
 M_Q = 0,1,2, \dots, 42 \\
 (44 * M_Q + 43 * N_Q) \\
 \text{must be calculated} \\
 \text{modulo 1118} \\
 \text{- Q - parity} \\
 \text{- Q - parity}
 \end{array}$$

The parity check matrix  $H_Q$  is:

Figure II.19 **Parity check matrix  $H_Q$**

$$H_Q = \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \\ a^{44} & a^{43} & \dots & a^1 & 1 \end{bmatrix}$$

where  $a$  is the primitive element of  $GF(2^8)$  defined above.

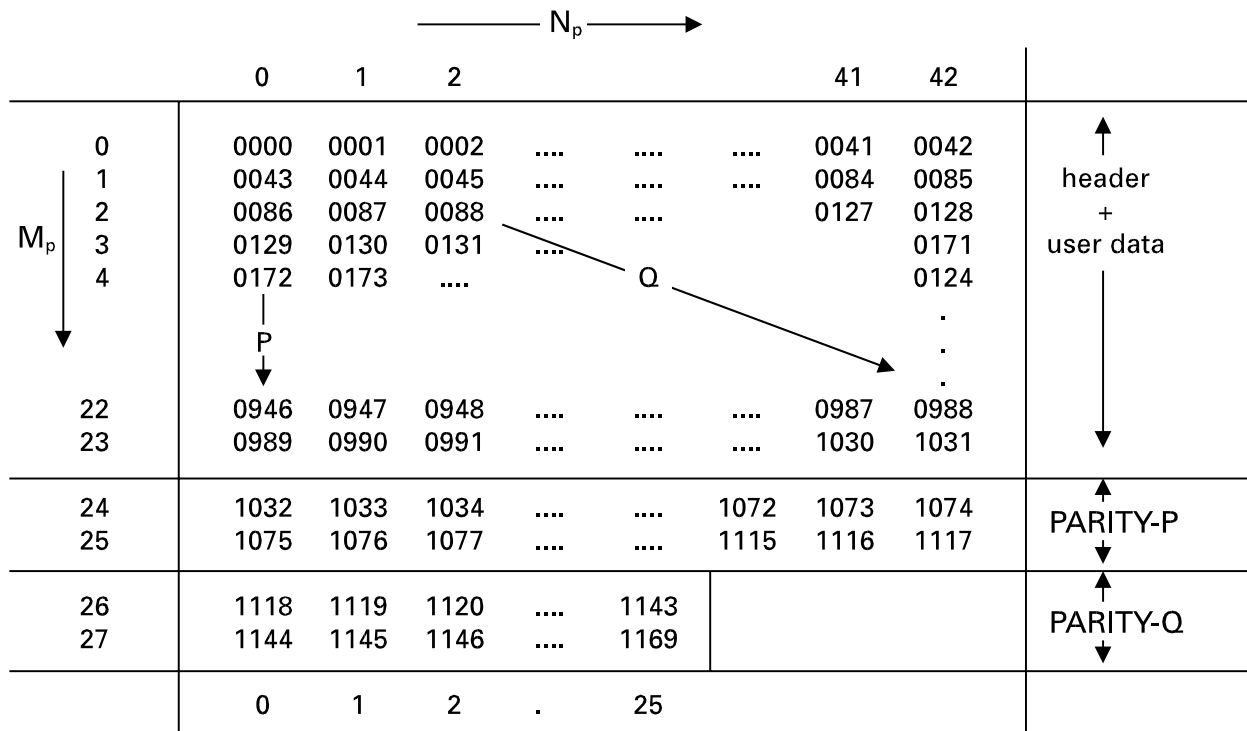
The quantity  $V_Q$  satisfies:

$$H_Q * V_Q = 0$$

II.4 CD-I Sector Specification

The interleave map for P-words and Q-words showing P-words as columns given in Figure II.20 below.

Figure II.20 Interleave map for P and Q-words



II.4 CD-I Sector Specification

---

The display of the symbols of the code in rectangular form showing Q-words as rows is given in Figure II.21.

Figure II.21 Rectangular form display of code symbols showing Q-words as rows

		————— $M_o$ —————>									
		0	1	2	.	.	40	41	42	Q0	Q1
$N_o$	0	0000	0044	0088	....	....	0642	0686	0730	1118	1144
	1	0043	0087	0131	....	....	0685	0729	0773	1119	1145
	2	0086	0130	0174	....	....	0728	0772	0816	1120	1146
	3	0129	0173	0217	....	....	0771	0815	0859	1121	1147
	4	0172	0216	0260	....	....	0814	0858	0902	1122	1148
		.									.
		.									.
		.									.
	22	0946	0990	1034	....	....	0470	0514	0558	1140	1166
	23	0989	1033	1077	....	....	0513	0557	0601	1141	1167
24	1032	1076	0002	....	....	0556	0600	0644	1142	1168	
25	1075	0001	0045	....	....	0599	0643	0687	1143	1169	

II.4 CD-I Sector Specification

---

**4.8 Form 2 Sector****4.8.1 General Layout**

A Form 2 sector consists of 2352 bytes. Each byte in the sector can be identified by  $B_i$  where  $i = 0$  to 2351.

The layout of a Form 2 sector is given in Figure II.22.

Figure II.22 **Layout of a Mode 2 Form 2 sector**

Byte	Meaning
$B_0$ to $B_{15}$	the synchronization and header fields
$B_{16}$ to $B_{23}$	the subheader field
$B_{24}$ to $B_{2347}$	2324 bytes of data whose content depends on the sector type
$B_{2348}$ to $B_{2351}$	4 bytes reserved for quality control during the CD-I disc production process

**4.8.2 Reserved Field**

The reserved field ( $B_{2348}$  to  $B_{2351}$ ) contains 4 bytes that are reserved for quality control during the CD-I disc production process. These bytes are ignored by the CD-I system. It is **recommended** that the same EDC algorithm should be used here as is used for Form 1 sectors. If this algorithm is not used, then the reserved bytes are set to 0.

## II.4 CD-I Sector Specification

---

### 4.9 Empty and Message Sectors

#### 4.9.1 Empty Sector

An empty sector may be Form 1 or Form 2 and does not contain any CD-I data. They may be used in the lead-in and/or lead-out areas. In the program area empty sectors can be used to fill up file space particularly for real-time files.

The subheader of an Empty sector has the following restrictions:

- (1) Channel number must be zero.
- (2) The Video, Audio and Data bits in the submode byte must be zero.
- (3) The Coding Information byte must be zero.

It is **recommended** that empty sectors are Form 2 and that the data bytes are zero.

#### 4.9.2 Message Sector

The data field of a message sector is in Form 2 and consists of 2324 bytes which do not contain any actual data for the CD-I system. The message sector is designed to provide a caution for the user of a CD-DA player that has no TOC decoder. The caution should warn the user to lower the volume and advance to the CD-DA tracks.

The message data must be encoded as CD-DA audio data in the mode 2 Form 2 message sector. As such, it is encoded after scrambling as actual CD-DA audio data so that it may be audible on normal CD-DA players. It is recommended that the warning be in English as well as any other language that may be appropriate. Figure II.23 gives an example of the message which might be contained in message sectors. All bits of the subheader bytes of a message sector except the Form bit are set to zero.

Figure II.23 **Example of a message**

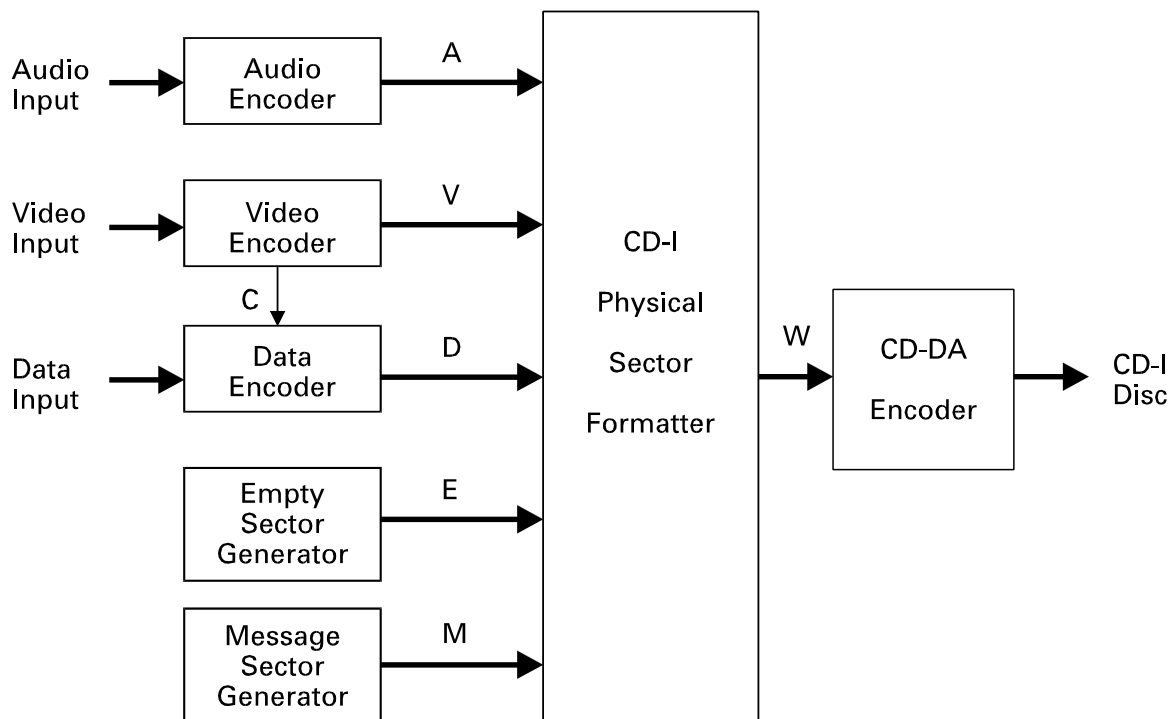
Message Example
"Please lower the volume temporarily and advance to the next track."

II.5 CD-I Encoder Model

II.5 CD-I Encoder Model

5.1 CD-I Disc Encoder Model

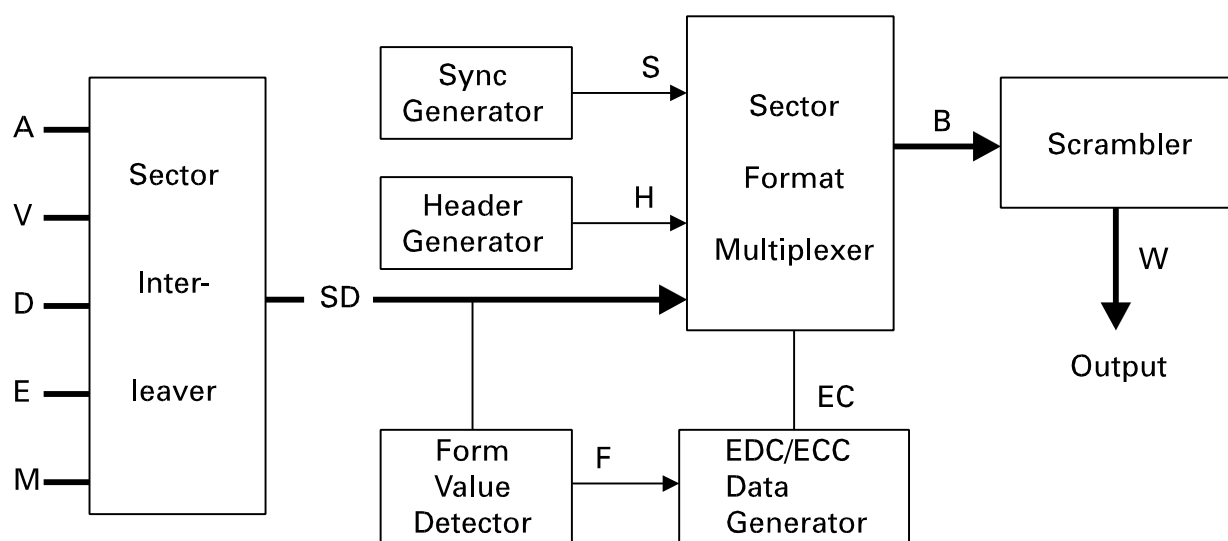
Figure II.24 CD-I Disc Encoder Model



- A = Audio sector
- V = Video sector
- D = Data sector
- E = Empty sector
- C = Video control data
- M = Message sector
- W = CD-I sector

## 5.2 CD-I Physical Sector Formatter

Figure II.25 CD-I Physical Sector Formatter



A = Audio sector data

V = Video sector data

D = Data sector data

E = Empty sector data

M = Message sector data

S = Synchronisation data

H = Header data

SD = Sector data

EC = Error detection and correction data

F = Form value

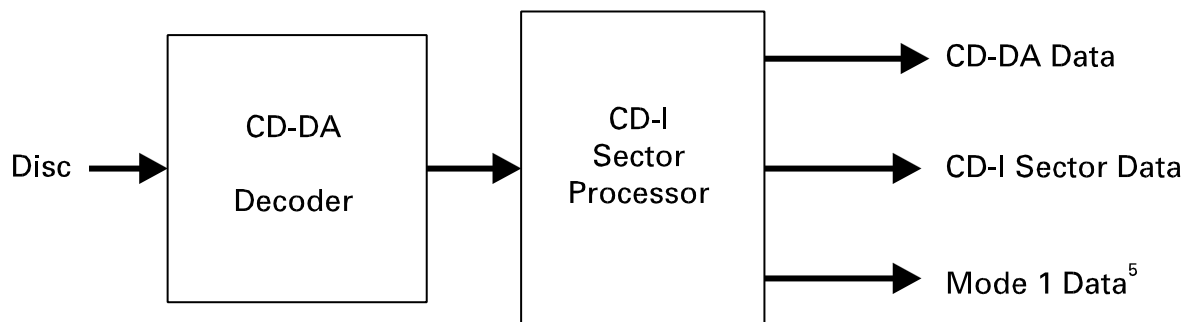
B = Serial bit stream of CD-I sector data

W = 16-bit word stream of CD-I sector data to CD-DA encoder



II.6 CD-I Decoder Model

---

II.6 **CD-I Decoder Model**6.1 **CD-I Decoder Model**Figure II.26 **CD-I Decoder Model**

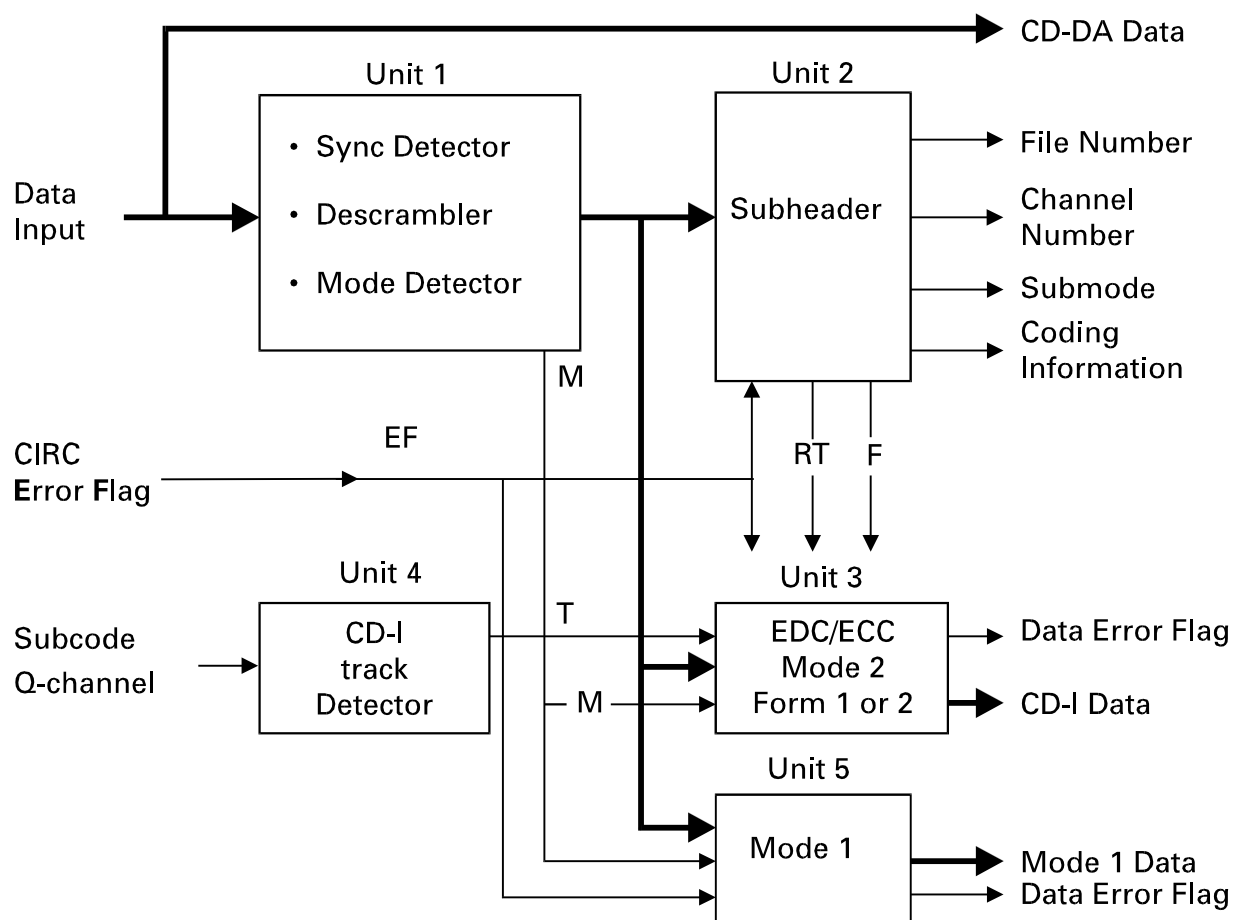
---

<sup>5</sup> A sector with the mode byte equal to one. For further details see VIII.2.3.

## II.6 CD-I Decoder Model

## 6.2 CD-I Sector Processor

Figure II.27 CD-I Sector Processor



M = Mode (1 or 2)  
 RT = Real-time Flag of Submode Byte  
 F = Form (1 or 2)  
 EF = Set if data is unreliable  
 T = Set if data track is a CD-I track

For CIRC see CD-DA specification pp 27 to 37.

For Subcode Q channel see CD-DA specification page 41.

II.6 CD-I Decoder Model

---

**Unit 1**

The Unit 1 processing is only valid for a data track.

- The synchronization detector is used for all timing needed in the sector processor.
- The descrambler processes the input data stream into sector data bytes ( $B_{12}$  to  $B_{2351}$ ).
- The mode detector gives the mode value for the data sector from the header field.

**Unit 2**

In Unit 2 the error flag from CIRC is used to select the file number, channel number, submode and coding information bytes from the subheader field ( $B_{16}$  to  $B_{23}$ ) that are reliable. The Real-time flag and the Form value in the submode byte are passed to Unit 3.

**Unit 3**

For Form 1 sectors the EDC and ECC data field is used to correct any erroneous data in the data field. In the case of a Form 2 sector no processing is required. For concealment purposes, a data error flag must be generated. If this error flag is set to one, an additional table of error flags per byte or word (depending on the implementation) is available. The format of these tables is given in VII.2.2.3.2

**Unit 4:**

Unit 4 has to detect if the disc is a CD-I disc or not. This can be done by using the information in the TOC (Subcode Q-channel).

The Unit 2 and Unit 3 processing are only valid for CD-I data tracks.

**Unit 5:**

For Mode 1 sectors, the EDC and ECC data field is used to correct any error in the data field.

A data error flag is set if error correction was not possible (see VII.2.2.3.4).

### 6.3 **Byte Order : After the Sector Processor**

The byte and bit ordering after the sector processor is as given in I.3.

Table of Contents

---

**Chapter III Data Retrieval Structure**

	<b>Page</b>
III.1 General	III-1
1.1 Application file names	III-3
III.2 Disc Label	III-4
2.1 General Structure of the Disc Label	III-4
2.2 Detailed Specification of the File Structure Volume Descriptor Record	III-5
2.3 Detailed Specification of a Terminator Record	III-14
2.4 Location of a Disc Label	III-15
2.5 Sector Subheader Format of Disc Label	III-16
2.6 Location of Data Track and Music Tracks	III-17
III.3 Disc Directory	III-18
3.1 General	III-18
3.2 Directory Structure	III-19
3.3 Directory Search Method	III-24
3.4 Sector Subheader Format	III-27
III.4 Files	III-28
4.1 General	III-28
4.2 Standard Files	III-29
4.3 Real-time Files	III-30
4.4 Interleaving Files	III-31
4.4.1 General	III-31
4.4.2 File Number	III-31

This page is intentionally left blank

List of Illustrations

---

<b>Fig. no.</b>	<b>Caption</b>	<b>Page</b>
III.1	Structure of a File Structure Volume Descriptor	III-6
III.2	Coding of bits in Volume flags held	III-8
III.3	Example of a standard File Structure Volume Descriptor	III-11
III.6	Structure of a Terminator record format	III-14
III.7	Example of a Terminator record	III-14
III.8	Structure of a directory record	III-20
III-9	Structure of Creation date field	III-21
III-10	Structure of File flags field	III-21
III.11	Structure of File Attribute field	III-22
III-12	Example of a directory record	III-23
III.13	Structure of an entry in the Path Table	III-24
III.14	Example directory structure	III-25
III.15	Entries in the Path Table for the Directory in Fig. III.14	III-26
III.16	Recommended files contained in a Root Directory	III-29
III.17	File Number Values	III-31

This page is intentionally left blank



## Chapter III Data Retrieval Structure

### III.1 General

This chapter contains the format of data structures used by the CD file manager to retrieve named files from a CD-I disc. It contains a specification for the data organization, Disc Label, and directory record structures.

From an access point of view, the disc consists of directly addressable **sectors**, numbered consecutively (beginning at zero). The main channel address of **block** zero is taken to be exactly 00 minutes 02 seconds and 00 sector number. The disc logical format specifies the contents of these sectors.

#### Definitions for this chapter:

- (1) The word sector is used to designate a CD-I **physical sector**. The data part of a **logical sector** is referred to as a **block**.
- (2) Block addresses are 32-bit integers. They are converted into absolute disc addresses (e.g., minutes, seconds, and sectors).
- (3) **BP** indicates Byte Position within a table, starting at 1.

#### Rules for construction of file names

Any file name length is limited to 28 bytes.

When the CD-I default character set (ISO 8859-1) is used the following rules must be followed:

A file name can contain any number of upper or lower case letters, numbers or special characters (as listed below). While the file name may begin with any of the following characters or digits, the file name must contain at least one letter or number. Within these limitations, it can contain any combinations of the following:

- upper case letter: A - Z
- lower case letter: a - z
- decimal digits: 0 - 9
- underscore: \_\_\_\_\_
- period: .
- dollar sign: \$

File names may not contain spaces.

III.1 General

---

Here are some examples of legal names:

raw.data.2	project_review_backup
X6809	\$Ship.Dir
.login	12345

Here are some examples of illegal names:

Max*min	(* is not a legal character)
open orders	(Name cannot contain a space)
this.name.obviously.has.more.than.28.characters (too long)	

III.1 General

---

**1.1 Application file names**

It is recommended that applications use module names that start with "cdi\_" (e.g. cdi\_myprog). This will help prevent conflicts with manufacturer dependent or standard CD-RTOS modules.

III.2 Disc Label

---

**III.2 Disc Label**

**2.1 General Structure of the Disc Label**

There are 2 record types in the CD-I Disc Label:

- (1) **File Structure Volume Descriptor Record.** Every CD-I disc contains at least one File Structure Volume Descriptor record.
  
- (2) **Terminator Record.** At least one Terminator record is required on every CD-I disc. It is the last record in the Disc Label and signifies the end of the Disc Label.

## **2.2 Detailed Specification of the File Structure Volume Descriptor Record**

There can be one or more File Structure Volume Descriptor records. If the File Structure Volume Descriptor record for the player's default character set (see VII.3.1.4.1.3) is not found, then the first File Structure Volume Descriptor will be used.

Figure III.1 illustrates the format of a File Structure Volume Descriptor.

## III.2 Disc Label

Figure III.1 Structure of a File Structure Volume Descriptor

BP	Size in Bytes	Description
1	1	Disc Label Record Type
2	5	Volume Structure Standard ID
7	1	Volume Structure Version number
8	1	Volume flags
9	32	System identifier
41	32	Volume identifier
73	12	Reserved
85	4	Volume space size
89	32	Coded Character Set identifier
121	2	Reserved
123	2	Number of Volumes in Album
125	2	Reserved
127	2	Album Set Sequence number
129	2	Reserved
131	2	Logical Block size
133	4	Reserved
137	4	Path Table size
141	8	Reserved
149	4	Address of Path Table
153	38	Reserved
191	128	Album identifier
319	128	Publisher identifier
447	128	Data Preparer identifier
575	128	Application identifier
703	32	Copyright file name
735	5	Reserved
740	32	Abstract file name
772	5	Reserved
777	32	Bibliographic file name
809	5	Reserved
814	16	Creation date and time
830	1	Reserved
831	16	Modification date and time
847	1	Reserved
848	16	Expiration date and time
864	1	Reserved
865	16	Effective date and time
881	1	Reserved
882	1	File Structure Standard Version number
883	1	Reserved
884	512	Application use
1396	653	Reserved

### III.2 Disc Label

---

The fields in the File Structure Volume Descriptor have the following meaning:

**Disc Label Record Type:** This field always indicates one of the following:

- the Standard File Structure Volume Descriptor record with the byte set to 1,
- the Coded Character Set File Structure Volume Descriptor record with the byte set to 2.

All other values for this byte are reserved.

The standard File Structure Volume Descriptor specifies a file structure with all character strings recorded using the ISO 646 (IRV) character set.

A Coded Character Set File Structure Volume Descriptor is used to describe a file structure which uses a character set other than the standard ISO 646 character set for the following descriptor fields:

(1) In the File Structure Volume Descriptor

- |                           |                            |
|---------------------------|----------------------------|
| - System identifier       | - Application identifier   |
| - Volume identifier       | - Data Preparer identifier |
| - Album identifier        | - Copyright file name      |
| - Publisher identifier    | - Abstract file name       |
| - Bibliographic file name |                            |

(2) File names in each directory record of the file structure.

(3) Directory file names in each Path Table Entry of the file structure.

A Coded Character Set file structure describes the same file system as the standard file structure, but the character strings in the above descriptor fields are encoded in an alternative character set identified by the Coded Character Set identifier.

Shifted JIS Kanji is specified by a Disc Label Record Type of two, a Volume flag of one, and the ISO 2022 escape sequence for standard Kanji.

III.2 Disc Label

---

**Volume Structure Standard ID:** This name field represents the standard to which the disc file structure conforms. CD-I discs contain the string "CD-I" followed immediately with a space character.

**Volume Structure Version number:** The number in this field represents the version number of the standard to which the disc conforms. The current version is one.

**Volume flags:** The bits of this field are numbered from 0 to 7 starting with the least significant bit. They specify the characteristics of the volume as outlined below in Figure III.2.

Figure III.2 Coding of bits in Volume flags held

Bit Position	Characteristic
0	A value of zero indicates the coded character set identifier field specifies only an escape sequence registered according to ISO 2375. A value of one indicates the Coded Character Set identifier field specifies an escape sequence not registered according to ISO 2375.
1 - 7	All bits are 0 (reserved for future standardization)

**System identifier:** This field identifies the operating system which may use any fields described as reserved for the system. All CD-I discs must contain the string "CD-RTOS" padded on the right with space characters.

**Volume identifier:** This field contains the logical name of the CD-I disc padded on the right with space characters.

**Volume space size in blocks:** This field represents the total size of the usable portion of the disc in blocks (e.g., the highest addressable block to the operating system is the value of this field minus 1).



### III.2 Disc Label

---

**Coded Character Set identifier:** This field specifies one escape sequence according to the International Register of Coded Character Sets to be used with escape sequence standards for recording. The ESC character, which is the first character of all sequences, shall be omitted when recording this field.

**Number of Volumes in Album:** The total number of discs in the album to which this disc belongs is written in this field.

**Album Set Sequence number:** This number specifies the relative sequence number of this disc in the album to which it belongs. The first disc in the sequence will be the number 1.

**Logical Block size:** This field contains the size of a block (in bytes) as seen by the file system. For CD-I discs, the block size is always 2048 bytes.

**Path Table size:** This field contains the size in bytes of the system Path Table.

**Address of Path Table:** The block address of the first block of the system Path Table is kept in this field.

**Album identifier:** This field contains the name of the album to which this disc belongs and is padded on the right with space characters.

**Publisher identifier:** This field is used to identify the publisher of the disc (i.e., who specified the contents of the disc) and is padded on the right with space characters.

**Data Preparer identifier:** This field identifies the author of the contents of the disc and is padded on the right with space characters.

**Application identifier:** This specifies the path name of the application program to execute when the disc is first mounted. Section VII.1.2 outlines how this name is used in the startup sequence.

**Copyright file name:** This field optionally contains the name of a file that contains a copyright message. Any desired information may be encoded in this file. It is accessed in an application dependent manner. This field is padded on the right with space characters. If there is no name, all characters are space characters.

### III.2 Disc Label

---

**Abstract file name:** This field optionally contains the name of a file that contains an abstract message for the disc. Any desired information may be encoded in this file. It is accessed in an application dependent manner. This field is padded on the right with space characters. If there is no name, all characters are space characters.

**Bibliographic file name:** This field optionally contains the name of a file that contains bibliographic information about the disc. Any desired information may be encoded in this file. It is accessed in an application dependent manner. This field is padded on the right with space characters. If there is no name, all characters are space characters.

**Creation date and time:** This contains the date and time that the CD-I disc was originally mastered. The format of a 16-byte date and time field is recorded as a string of numerals "YYYYMMDDHHMMSSst" where tt means hundredths of a second. For example, 6:51 a.m. on September 6, 1954 would be recorded as "1954090606510000". No consideration is made for time zones. If a date is not used, it contains "0" characters.

**Modification date and time:** Recorded in this field are the date and time that the disc was last changed (e.g., re-mastered). The date is recorded in the same format as the creation date and time.

**Expiration date and time:** If the data on a disc is valid only for a limited time, this field indicates the time at which the disc becomes obsolete. The date is recorded in the same format as the creation date and time.

**Effective date and time:** If the data on a disc is not valid until a specific time, this field indicates the time at which the disc becomes useful. The date is recorded in the same format as the creation date and time.

**File Structure Standard Version number:** This number is used to indicate the revision number of the file structure standard to which the directory search files conform. It is set to one.

Figure III.3 is an example of a standard File Structure Volume Descriptor.

## III.2 Disc Label

Figure III-3 Example of a standard File Structure Volume Descriptor

BP	Contents	Description
1	1	Disc Label Record Type - standard SVD
2	"CD-I"	Volume Structure Standard ID
7	1	Volume Structure Version number
8	0	Volume flags - standard file structure
9	"CD-RTOS"	System identifier
41	"Games"	Volume identifier
73	0	Reserved
85	200000	Volume space size - 200,000 blocks used on this volume
89	0	Coded Character Set identifier - must be zero if standard FS
121	0	Reserved
123	1	Number of Volumes in Album
125	0	Reserved
127	1	Album Set Sequence number
129	0	Reserved
131	2048	Logical Block size
133	0	Reserved
137	2358	Path Table size
141	0	Reserved
149	2268	Address of Path Table
153	0	Reserved
191	"Games"	Album identifier
319	"RG Software"	Publisher identifier
447	"Larry Hobbs"	Data Preparer identifier
575	"Menu"	Application identifier
703	"Copyrightfile"	Copyright file name
735	0	Reserved
740	"Abstractfile"	Abstract file name
772	0	Reserved
777	"Bibliofile"	Bibliographic file name
809	0	Reserved
814	"1957100207340000"	Creation date and time
830	0	Reserved
831	"0000000000000000"	Modification date and time - not used if zero
847	0	Reserved

(continued)

III.2 Disc Label

---

(continued)

Figure III-3 **Example of a standard File Structure Volume Descriptor**

848	"0000000000000000"	Expiration date and time - not used if zero
864	0	Reserved
865	"0000000000000000"	Effective date and time - not used if zero
881	0	Reserved
882	1	File Structure Standard Version number
883	0	Reserved

This page is intentionally left blank

### 2.3 Detailed Specification of Terminator Record

The last record in the Disc Label must be a Terminator record. This record determines the end of the Disc Label. The format of the Terminator record is indicated in Figure III.6.

Figure III.6 **Structure of a Terminator record format**

BP	Size in bytes	Description
1	1	Disc Label Record Type
2	5	Volume Structure Standard ID
7	1	Volume Structure Version
8	2041	number Reserved

The fields in the record have the following meaning.

**Disc Label Record Type:** The value of 255 in this field signifies a Terminator record.

**Volume Structure Standard ID:** This name field represents the standard to which the disc file structure conforms. CD-I discs contain the string "CD-I", padded on the right with space characters.

**Volume Structure Version number:** The number in this field represents the version number of the standard to which the disc conforms. The current version is one.

Figure III.7 is an example of a Terminator record.

Figure III.7 **Example of a Terminator record**

BP	Contents	Description
1	255	Disc Label Record Type
2	"CD-I"	Volume Structure Standard ID
7	1	Volume Structure Version
8	0	number Reserved

III.2 Disc Label

---

**2.4 Location of Disc Label**

The Disc Label is located starting at block 16 of track 1 on a CD-I disc.

### 2.5 Sector Subheader Format of Disc Label

All sectors in the disc label have the subheader format which is given below.

- (1) The **File** number byte is zero
- (2) The **Channel** number bits are set to zero
- (3) The EOR and data bits in the **submode** byte are set to one and the rest are zero. Additionally the EOF bit in the **submode** byte of the last sector of the Disc Label is set to one.
- (4) The **Coding Information** byte is zero.



### 2.6 Location of Data Track and Music Tracks

CD-I data and programs (i.e. ADPCM sound, video data, text data, computer programs, and binary data) are contained in blocks after the message sectors which follow the Disc Label. If no CD-DA tracks (i.e., music tracks) are on a CD-I disc, then it is **recommended** that the CD-I data including the Disc Label and any application software as well as the message sectors are contained on one track.

The Disc Label must always be in track 1.

It should be noted that accessing of CD-DA data can be done within  $\pm 1$  second whilst accessing CD-I data can be done within the sector required.

III.3 Disc Directory

---

**III.3 Disc Directory**

**3.1 General**

On a CD-I disc there must be at least one file, organized as a directory, called the Root Directory. The Root Directory may be located anywhere on a disc. Its position is defined by an entry in the Path Table. It contains names and characteristics of files and optional subdirectories on the disc.

### III.3 Disc Directory

---

#### 3.2 Directory Structure

A directory is a file of variable length directory records. Each record is a file descriptor used to store common format and attribute information needed to access a file on the disc.

Directory files are used extensively by the compact disc file management system to locate other files on a CD-I disc. They may also be accessed by application programs to determine a disc's contents. The first record in any directory describes the directory itself. The directory name stored in the first directory record is a null byte. The second record describes the directory's parent directory file, except when the directory is the Root Directory, in which case the second record is a copy of the Root Directory record. The directory name stored in the second directory record is a byte which contains the value 1.

Every directory record must end in the sector in which it begins. Any unused bytes at the end of a sector of directory records must be set to null. Each directory record describes one file, as shown in Figure III.8.

Figure III.8 Structure of a directory record

BP	Size in bytes	Description
1	1	Record length
2	1	Extended Attribute record length
3	4	Reserved
7	4	File beginning LBN
11	4	Reserved
15	4	File size
19	6	Creation date
25	1	Reserved
26	1	File flags
27	2	Interleave
29	2	Reserved
31	2	Album Set Sequence number
33	1	File name size
34	(n)	File name
34+n	4	Owner ID
38+n	2	Attributes
40+n	2	Reserved
42+n	1	File number
43+n	1	Reserved
	43+n	Total

LBN = Logical **B**lock **N**umber

The fields in a directory record have the meaning given below.

**Record length:** This is the size in bytes of this directory record.

**Extended Attribute record length:** This is the number of blocks at the beginning of the file reserved for extended attribute information. The format of the extended attribute record is not defined and is reserved for application use.

**File beginning LBN:** This is the logical block number of the first block of the file.

**File size:** This is the size of the file in bytes (By convention the size of Form 2 sectors is considered as 2048 bytes by the system).

## III.3 Disc Directory

**Creation date:** This is the date and time on which the file was created or last modified. The field contains six bytes representing six unsigned numerical values in binary notation according to the following format.

Figure III-9 Structure of Creation date field

BP	Description
1	Number of years since 1900
2	Month of the year from 1 to 12
3	Day of the month from 1 to 31
4	Hour of the day from 0 to 23
5	Minute of the hour from 0 to 59
6	Second of the minute from 0 to 59

**File flags:** This field is bit encoded. Bit zero, when set to one, has the following meaning:

Figure III-10 Structure of File flags field

bit	
0	The file is hidden and does not usually appear in directory listings.
1-7	Reserved

**Interleave:** This contains two unsigned 8-bit integers indicating how the file is interleaved. The numbers represent a ratio between sectors that belong to the file and those that do not. As an example, an interleave value of 1:3 would represent a file sector map as follows.

\* - - - \* - - - \* ...

(\* means a sector is part of the file and - means it is unrelated to the file.)

For non-interleaved files both bytes must be set to zero.

**Album Set Sequence number:** The Album Set Sequence number of the disc which contains this file is recorded in this field. If the file resides on this disc this field is zero. The use and definition of the Album Set Sequence number is the responsibility of the content provider.

## III.3 Disc Directory

**File name size:** This field is used to record the number of characters in the file name field. If the length of the file name is even, a null padding byte is inserted on the end of the name to make the owner ID field begin on an even offset. The padding byte is not included in the file name size field.

**File name:** This field contains the name of the file described by this directory record.

**Owner ID:** This field contains the user identification number of the creator of this file. The owner's group ID is contained in the most significant two bytes and the owner's user ID is contained in the least significant two bytes.

**Attributes:** This field contains the file attributes and permissions, bit encoded. It indicates how the file is accessed and who (owner ID) may access it. If the owner bit is set to one then a user with the same user ID as the owner ID of this file may access the file. If the group bit is set to one then any user with the same group ID as the owner's group ID may access the file. If the world bit is set to one then any user may access the file. There is also a bit which if set to one indicates if a file is a CD-DA file. The format of the attributes is outlined in Figure III.11. If the directory bit is set to one then this file is a directory.

Figure III.11 Structure of File Attribute field

Bit Number	Contents
0	Owner read
1	Reserved
2	Owner execute
3	Reserved
4	Group read
5	Reserved
6	Group execute
7	Reserved
8	World read
9	Reserved
10	World execute
11-13	Reserved
14	CD-DA file
15	Directory

## III.3 Disc Directory

**File number:** This number is recorded as the file identifier in the subheader of each sector belonging to a Mode 2 file. It is used to select sectors that belong to the file.

Figure III.12 is an example of a directory record.

Figure III.12 **Example of a directory record**

BP	Contents	Description
1	48	Record length
2	0	Extended Attribute record length
3	0	Reserved - must be zero
7	29	File beginning LBN
11	0	Reserved - must be zero
15	1000000	File size in bytes
19	\$39 08 18 0A 20 03	Creation date (24 August 1957, 10:32:03)
25	0	Reserved
26	0	File flags - not hidden
27	0204	Interleave (2:4)
29	0	Reserved
31	0	Album Set Sequence number
33	5	File name size
34	"GAMES"	File name
39	\$000A0005	Owner ID - group = 10, user = 5
43	\$0055	Attribute - read and execute for owner and group
45	0	Reserved
47	2	File number
48	0	Reserved

## III.3 Disc Directory

## 3.3 Directory Search Method

As part of the CD-I mastering process, a Path Table index is created to describe the entire directory structure on disc. This index is recorded in the disc Root Directory as a normal CD-I file. The Path Table is loaded into memory when a disc is mounted or first accessed. It permits any file on disc to be opened using only one seek.

The Path Table is organized as a breadth first traversal (see Fig. III.12 and III.13) of the disc directory structure. When a file is opened the Path Table is sequentially searched (in RAM) to locate the directory that contains the record describing the file. This search yields the block number of the beginning of the directory file. A seek is then made to this block and records are searched sequentially until the record for the requested file is found or it is determined that the file does not exist.

The Path Table contains one variable length entry for each directory and subdirectory on a disc. Each entry is organized as shown in Figure III.13.

Figure III.13 Structure of an entry in the Path Table

BP	Size in bytes	Description
1	1	Name size
2	1	Extended Attribute record length
3	4	Directory block address
7	2	Parent Directory number
9	n	Directory file name

The fields have the following meaning:

**Name size:** The length of the directory file name string in this path table entry.

**Extended Attribute record length:** This is the length of the Extended Attribute record.

**Directory block address:** This field contains the beginning logical block number (LBN) of the directory file on disc.

**Parent Directory number:** This is the number (relative to the beginning of the Path Table) of this directory's parent.



III.3 Disc Directory

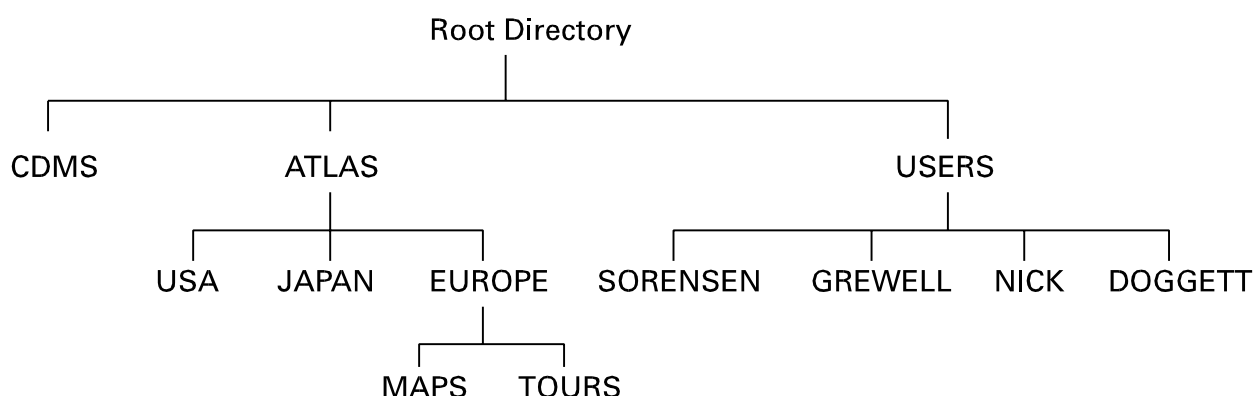
---

**Directory file name:** This variable length field is used to store the actual text representing the name of the directory. If the length of the file name is odd, a null padding byte is added to make the size of the Path Table record even. The padding byte is not included in the name size field.

If there is more than one directory on the disc, the first entry of the Path Table describes the Root Directory with a parent directory of 1 and directory file name = 0.

Figure III.14 depicts an example directory structure and the contents of the Path Table that would be used to describe it to the CD-I File Manager.

Figure III.14 Example directory structure



The Path Table used to describe the above directory structure would contain the entries given in Figure III.15.

## III.3 Disc Directory

Figure III.15 Entries in the path table for the directory in Fig. III.14

Relative Position	Directory Block Address	Extended Attribute Record Length	Parent Directory Number	Name Size	Directory File Name
1	XXXXXXX	0	1	1	0 (Root)
2	XXXXXXX	0	1	5	"ATLAS"
3	XXXXXXX	0	1	4	"CMDS"
4	XXXXXXX	0	1	5	"USERS"
5	XXXXXXX	0	2	6	"EUROPE"
6	XXXXXXX	0	2	5	"JAPAN"
7	XXXXXXX	0	2	3	"USA"
8	XXXXXXX	0	4	7	"DOGGETT"
9	XXXXXXX	0	4	7	"GREWELL"
10	XXXXXXX	0	4	4	"NICK"
11	XXXXXXX	0	4	8	"SORENSEN"
12	XXXXXXX	0	5	4	"MAPS"
13	XXXXXXX	0	5	5	"TOURS"

As illustrated above, directory files on the same level (e.g., with the same Parent Directory) are placed in the Path Table in binary ascending order of the file name.

III.3 Disc Directory

---

**3.4 Sector Subheader Format**

All sectors in the directory and the path table have the subheader format which is given below.

1. The File number byte is zero
2. The Channel number bits are set to zero
3. The data bit in the submode byte is set to one and the rest are zero.

Additionally the EOR and the EOF bits in the submode byte of the last sector of the Directory and the Path Table are set to one.

4. The Coding Information byte is zero.

### III.4 Files

---

#### III.4 Files

##### 4.1 General

A CD-I file is composed of a set of (logically) contiguous blocks identified by a unique name in a directory. These blocks are numbered starting at zero, which is the number of the first block of the file. Files are made up of blocks of either 2048 or 2324 bytes (or both). They can contain any combination of audio, video, or computer data.

Random access to files at the byte level is available through system calls provided by the Compact Disc File manager.

An attribute bit in the directory record of a file may be set to one to indicate the file is an audio track, recorded in CD-DA format. This type of file can only be played through a CD-DA decoder in the Audio Processing Unit (see IV 5.2). The CD-DA data in the file is not otherwise accessible by application programs.

III.4 Files

---

**4.2 Standard Files**

By convention, every CD-I disc contains a small number of files in its Root Directory. The names and contents of recommended files appear in Figure III.16.

Figure III.16 **Recommended files contained in a Root Directory**

Name	Contents
"path_tbl"	This file is used to record an image of the Path Table index as specified in III.3.3.
"album"	A catalogue of related CD-I discs is recorded in this file. The information in this file may be used by an application to instruct a user to locate and mount a particular disc in a multi-album set.
"author"	This file is used to identify the CD-I content developer i.e. data preparer. Any desired information may be recorded in the file.
"manufacturer"	This file is used to identify the manufacturer of the CD-I disc. Any desired information may be recorded in the file.

### 4.3 Real-time Files

A real-time file is a file containing real-time records. A real-time record is a logical record in a CD-I file containing audio, video, and/or computer data that must be retrieved from a CD-I disc at a precise rate in order to produce an aesthetic sound or image. In particular, audio data within a real-time record, whether from disc or RAM, must arrive at the audio processor in exact intervals in order to be perceived as a realistic sound. For example, the audio blocks in a real-time record should be considered a clock upon whose timing the other elements in a real-time record must be synchronized.

Individual blocks of real-time records can contain audio, video, or ordinary computer data, but these different types of data cannot be mixed within a single block. The Compact Disc File Manager (see VII.2.2) does not restrict the order of audio, video, or data blocks. This order is determined according to the requirements of the application. When a real-time record is read, blocks are grouped together or directly routed to the appropriate hardware (e.g., audio data).

Data blocks that contain computer data are always read into memory to be processed by an application program.

The last physical sector belonging to a real-time record is identified by having an EOR bit set to one in its sector subheader submode byte.

III.4 Files

---

**4.4 Interleaving Data and Files****4.4.1 General**

The audio blocks in a real-time file from a CD-I disc must arrive to the audio processor at a very precise rate. The rotational latency of CD drives makes it impossible to achieve the required data rate by simply reading an audio block, pausing for a predetermined time, and then reading the next block. Instead, audio blocks must be physically interspaced at intervals which depend on the quality of sound reproduction.

The blocks between consecutive audio blocks can be used in a variety of ways. Video (i.e., stills) or text data does not necessarily have such a critical timing and can be placed within the gaps. Also, a number of mutually exclusive audio channels can be interleaved within a real-time record to provide different sound tracks, for example, each in a different language.

It is also possible to physically interleave files (see III.4.4.2) according to a fixed repeating pattern.

Any combination of these techniques may be used depending on the requirements of an application. Information recorded in the subheader of each block is used to determine which blocks are to be played. Directory files must not be interleaved.

**4.4.2 File Number**

The file number is used to identify all sectors that belong to one file. A given file may be physically interleaved with other files on the disc. All sectors of a logical file have the same value in the file number byte. The file number can be used to select sectors that belong to the same logical file and to reject any others. The file number with a byte value of 0 is only used for a file that will be read consecutively. No interleaving is possible with file 0. Files 1 to 255 (see Figure III-17) may be read consecutively or may be interleaved with other files.

Figure III.17 **File Number Values**

Byte Value	File Number
0000 0000	0 (consecutive sector reading only)
0000 0001	1
0000 0010	2
:	:
1111 1111	255

This page is intentionally left blank



## Table of Contents

<b>Chapter IV Audio Data Representations</b>		<b>Page</b>
IV.1	General Audio Encoding	IV-1
IV.2	Sound Quality Levels	IV-3
IV.3	Audio Sector Data Format	IV-4
	3.1 General	IV-4
	3.2 Subheader Bytes	IV-5
	3.2.1 File Number	IV-5
	3.2.2 Channel Number	IV-5
	3.2.3 Submode	IV-5
	3.2.4 Coding Information	IV-6
	3.3 Audio Block	IV-7
	3.4 Sound Group	IV-8
	3.5 Level A Audio	IV-9
	3.6 Level B and Level C Audio	IV-11
	3.7 Audio Sector Interleaving	IV-13
IV.4	Audio Data Encoding	IV-14
	4.1 General	IV-14
	4.2 ADPCM Encoder	IV-16
	4.3 Sound Parameters	IV-18
IV.5	ADPCM Decoder	IV-19
	5.1 Audio Decoder Model	IV-19
	5.2 Audio Processing Unit	IV-20
	5.3 ADPCM Decoder	IV-21
	5.4 Soundmap Unit	IV-23
	5.5 Microprocessing Unit	IV-24
IV.6	Audio Data Rearrangements	IV-25
	6.1 Non Real-time Rearrangements	IV-25
	6.2 Real-time Rearrangements	IV-26
	6.3 Audio Mixing Control Unit	IV-27
IV.7	Extended Audio Playing Time	IV-28

This page is intentionally left blank

List of Illustrations

---

<b>Fig. no.</b>	<b>Caption</b>	<b>Page</b>
IV.1	General encoding for ADPCM mono or stereo sound	IV-2
IV.2	Overview of sound quality levels	IV-3
IV.3	Encoding for the submode byte of an Audio sector	IV-5
IV.4	Format of Audio Coding Information byte	IV-6
IV.5	Encoded sequential order for sound groups	IV-7
IV.6	Layout of a sound group	IV-8
IV.7	Format of Sound Parameter Byte	IV-8
IV.8	Sound parameter layout for Level A audio	IV-10
IV.9	Sound data layout for level A audio	IV-10
IV.10	Sound parameter layout for audio levels B and C	IV-12
IV.11	Sound data layouts for audio levels B and C	IV-12
IV.12	Audio sector interleaving	IV-13
IV.13	General configuration of the ADPCM encoder	IV-15
IV.14	Predictor unit	IV-16
IV.15	Noise shaper unit	IV-17
IV.16	Gain values for sound parameter filters	IV-18
IV.17	Range values for sound quality levels	IV-18
IV.18	Audio decoder model	IV-19
IV.19	ADPCM decoder	IV-21

This page is intentionally left blank

List of Illustrations

---

<b>Fig. no.</b>	<b>Caption</b>	<b>Page</b>
IV.20a	Soundmaps and Real Time Audio Rearrangements	IV-26
IV.20b	Real Time Audio Rearrangements Timing	IV-26
IV.21	Audio mixing control unit	IV-27
IV.22	Example of extended audio playing time using level C audio	IV-28

This page is intentionally left blank

## **Chapter IV Audio Data Representations**

### **IV.1 General Audio Encoding**

This chapter defines the format and encoding of audio data for a CD-I disc.

Digitally recorded sound in mono or stereo can be encoded as shown in Figure IV.1. The required sound quality level to be encoded has to be set for the sample rate converter and the ADPCM encoder, giving the sampling frequency and number of bits per sample on the disc. The ADPCM encoded audio data is then formatted into an audio sector.

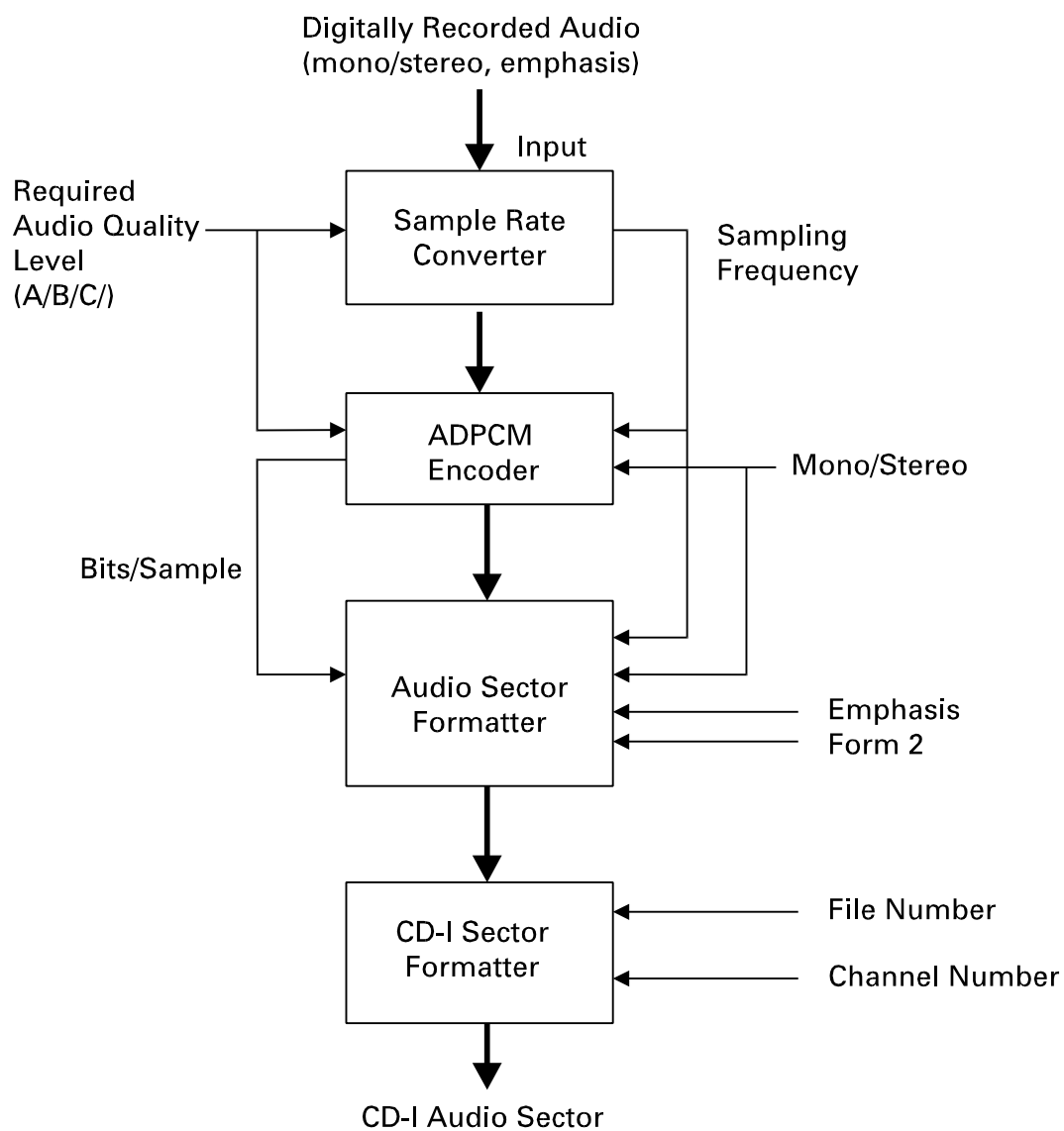
This chapter also defines how the encoded data from a CD-I disc is decoded back to audible sound.

Abbreviations used in this chapter have the following meanings:

ADPCM : Adaptive Delta Pulse Code Modulation  
PCM : Pulse Code Modulation  
CD-DA : Compact Disc - Digital Audio  
S/N : Signal to noise ratio

Other abbreviations used are defined in the text.

Figure IV.1 **General encoding for ADPCM mono or stereo sound**





IV.2 Sound Quality Levels

---

**IV.2 Sound Quality Levels**

Audio data from more than one audio source can be encoded in a CD-I track. Each audio source will be identified with an audio channel number in the subheader (see IV.3) of the CD-I sector.

The encoded sound quality level is dependent on the sampling frequency (fs) and the number of bits used per sample (b). The bit rate coming from the CD-I player is constant, but the maximum available channel numbers (N) can vary (see Figure IV.2 below). From the basic audio level parameters one can derive the maximum:

- audio bandwidth (BW) and
- channel numbers (N).

Figure IV.2 gives an overview of the sound quality levels

Figure IV.2 **Overview of sound quality levels**

Level	fs	b	BW	N	Stereo/ Mono
CD-DA	44.1 kHz	16	20 kHz	1	stereo
CD-I ADPCM					
Level A	37.8 kHz	8	17 kHz	2 4	stereo mono
Level B	37.8 kHz	4	17 kHz	4 8	stereo mono
Level C	18.9 kHz	4	8.5 kHz	8 16	stereo mono

IV.3 Audio Sector Data Format

---

**IV.3 Audio Sector Data Format**

**3.1 General**

The data field of an audio sector (see II.4.8) contains three fields:

- a subheader containing the file number, channel number, submode, and coding information bytes;
- an audio block data area containing 2304 bytes; and
- a padded field containing 20 bytes with value zero.

### 3.2 Subheader Bytes

#### 3.2.1 File Number

See III.4.4.2 for a definition of the file number.

#### 3.2.2 Channel Number

See Appendix II.1.2 for the general definition of channel numbers. It should be noted that for audio sectors, the channel number can only have the value of 0 to 15.

The audio channel selection for the ADPCM decoder is controlled by a separate 16-bit audio channel selection register. Any audio channels selected by the 32-bit channel selection register but not the 16-bit register are transferred to the host system along with the other (non-audio) channels. The audio channel selected by the 16-bit register is transferred to the audio processor.

#### 3.2.3 Submode

The submode byte has to be encoded as given in Figure IV.3 for an audio sector.

Figure IV.3 **Encoding for the submode byte of an Audio sector**

bit number	7	6	5	4	3	2	1	0
bit value	x	x	1	x	0	1	0	x

where x is defined in II.4.5.3.

IV.3 Audio Sector Data Format

---

**3.2.4 Coding Information**

The audio Coding Information byte is defined as follows:

Figure IV.4 **Format of Audio Coding Information byte**

0	Emphasis	Bits per Sample	Sampling Frequency	Mono/Stereo
bit 7	6	5	4 3	2 1 0

**Reserved**

Bit 7 = Set to zero

**Emphasis (see IV.4.2)**

Bit 6

0 = Emphasis off

1 = Emphasis on

**Number of Bits per Sample**

Bits 5-4

0 0 = 4 bits

0 1 = 8 bits

1 0 = Reserved

1 1 = Reserved

**Sampling Frequency**

Bits 3-2

0 0 = 37.8 kHz

0 1 = 18.9 kHz

1 0 = Reserved

1 1 = Reserved

**Mono/Stereo**

Bits 1-0

0 0 = Mono

0 1 = Stereo

1 0 = reserved

1 1 = reserved

**3.3 Audio Block**

The audio block field bytes  $B_{24}$  to  $B_{2327}$  of a CD-I audio sector consists of 2304 bytes. The audio block is further subdivided into 18 sound groups ( $SG_{00} \dots SG_{17}$ ) of 128 bytes each. The sound groups have to be encoded in sequential order (see Figure IV.5).

Figure IV.5 **Encoded sequential order for sound groups**

Audio Block	Sound Group
$B_{24}$ . . . $B_{151}$	$SG_{00}$
. . .	$SG_{xx}$
$B_{2200}$ . . $B_{2327}$	$SG_{17}$

IV.3 Audio Sector Data Format

---

**3.4 Sound Group**

A sound group is divided into two parts:

- Sound parameters (SP) : 16 bytes
- Sampled audio data : 112 bytes

The bytes  $S_i$  in the sound group (SG) are indexed per SG from 0 to 127. Figure IV.6 gives the layout of a sound group.

Figure IV.6 **Layout of a Sound Group**

Sound Group Byte numbers	Meaning
$S_0$ . $S_{15}$	Sound Parameter Bytes
$S_{16}$ . . . $S_{127}$	Sample Audio  Data Bytes

The sound parameter bytes each contain the two sound parameters, i.e., the range (R) and filter (F) (see IV.4.3).

The sound parameter (SP) has the value given by:

$$SP = 16 * F + R$$

where the range (R) and filter (F) positions in the sound parameter byte are shown in Figure IV.7.

Figure IV.7 **Format of Sound Parameter byte**

Sound Parameter Byte SP	
Filter (F)	Range (R)
bits 7 to 4	bits 3 to 0

IV.3 Audio Sector Data Format

---

**3.5 Level A Audio**

In level A audio, a sound group (SG) contains four sound units ( $SU^j$ ) with  $j = 0$  to  $3$ . A sound unit consists of 4 identical sound parameter (SP) bytes of 8 bits and 28 sound data bytes.

The sound parameters ( $SP^j$ ) for the sound units ( $SU^j$ ) are encoded in  $S_0$  to  $S_{15}$  (see IV.3.4) where  $j$  is the sound unit index. Here,

$$S_i = SP^j$$

where  $j = i \bmod 4$  and  
 $i$  = the byte number index in sound group (see Figure IV.7)

The 8-bit sound data bytes ( $SB_k^j$ ) are encoded in  $S_{16}$  to  $S_{127}$  where  $k$  is the sequential sampling order indexed from 0 to 27. Here,

$$S_i = SB_k^j$$

where  $i = 16 + j + 4 * k$  (see Figure IV.9)

In mono, the sound units (SU) are encoded sequentially i.e.  $SU^0$ ,  $SU^1$ ,  $SU^2$  and  $SU^3$ .

In stereo, the left signal is  $SU^0$  and  $SU^2$  and the right signal is  $SU^1$  and  $SU^3$ . The sound units (SU) are encoded in sequential pairs, i.e.  $SU^0$  and  $SU^1$  are encoded together, followed by the pair  $SU^2$  and  $SU^3$ .

Figure IV.8 shows the sound parameter layout for level A audio whilst Figure IV.9 gives the sound data layout for this audio level.

Figure IV.8 **Sound parameter layout for audio levels B and C**

Sound Group Byte Number (=i)	Sound Unit Number (=j)
0	0
1	1
2	2
3	3
4	0
5	1
6	2
7	3
8	0
9	1
10	2
11	3
12	0
13	1
14	2
15	3

Figure IV.9 **Sound data layouts for audio levels B and C**

Sound Group Byte Number (=i)	Sound Unit Number (=j)	Sound Sample Number (=k)
16	0	0
17	1	0
18	2	0
19	3	0
20	0	1
21	1	1
.	.	.
.	.	.
126	2	27
127	3	27



IV.3 Audio Sector Data Format

---

**3.6 Level B and Level C Audio**

In levels B and C audio, a sound group (SG) consists of 8 sound units ( $SU^j$ ) where  $j = 0$  to 7. A sound unit consists of 2 identical sound parameter bytes of 8 bits and 28 sound data nibbles ( $SD_k^j$ ) of 4 bits.

The sound parameters ( $SP^j$ ) for the sound units ( $SU^j$ ) are encoded in  $S_0$  to  $S_{15}$  (see IV.3.4) and  $j$  is the sound unit index. Here,

$$S_i = SP^j$$

where the relation between  $i$  and  $j$  is given by Figure IV.10.

The 4 bit sound data ( $SD_k^j$ ) from two sound units are combined into a sound byte ( $SB_k^l$ ) where  $k$  is the sequential sampling order indexed from 0 to 27 in the following manner:

$$SB_k^l = SD_k^j + 16 * SD_k^{j+1} \quad (\text{see Figure IV.11})$$

$$\begin{aligned} \text{where } j &= 2 * l \text{ and} \\ l &= 0 \text{ to } 3 \end{aligned}$$

These sound bytes ( $SB_k^l$ ) are encoded in  $S_{16}$  to  $S_{127}$ . Here,

$$S_i = SB_k^l$$

$$\text{where } i = 16 + l + 4 * k$$

In mono, the sound units are encoded sequentially i.e.  $SU^0, SU^1, SU^2 \dots SU^7$ . In stereo, the left signal is given by  $SU^0, SU^2, SU^4$ , and  $SU^6$  and the right signal is given by  $SU^1, SU^3, SU^5$  and  $SU^7$ . The sound units are encoded in sequential pairs i.e.  $SU^n$  and  $SU^{n+1}$  are encoded together where  $n = 0, 2, 4$  and 6.

Figure IV.10 shows the sound parameter layout for levels B and C audio and Figure IV.11 gives the sound data layout for these audio levels.

IV.3 Audio Sector Data Format

---

Figure IV.10 **Sound Parameter layout for audio levels B and C**

Sound Group Byte Number (=i)	Sound Unit Number (=j)
0	0
1	1
2	2
3	3
4	0
5	1
6	2
7	3
8	4
9	5
10	6
11	7
12	4
13	5
14	6
15	7

Figure IV.11 **Sound data layouts for audio levels B and C**

Sound Group Byte Number (=i)	Sound Unit Number (=j)	Sound Sample Number (=k)
16	1 and 0	0
17	3 and 2	0
18	5 and 4	0
19	7 and 6	0
20	1 and 0	1
21	3 and 2	1
.	.	.
.	.	.
126	5 and 4	27
127	7 and 6	27

**3.7 Audio Sector Interleaving**

Figure IV.12 **Audio sector interleaving**

Relative Sector Number																	
Level	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A stereo	*		*		*		*		*		*		*		*		*
A mono	*				*				*				*				*
B stereo	*				*				*				*				*
B mono	*								*								*
C stereo	*								*								*
C mono	*																*

Figure IV.12 shows the possibilities for audio sector interleaving in the case of a real-time bit being set to one. The interleave factors are fixed for each quality level and refer to the audio sectors with the same file number and channel number; the other audio sectors must have another file number or channel number.

The audio sector interleaving is not necessarily applicable if the real-time bit is zero for audio sectors. It should be noted that non-real-time audio data must be sent to a soundmap.

## IV.4 Audio Data Encoding

---

### IV.4 Audio Data Encoding

#### 4.1 General

The principal feature of the ADPCM system (see Figure IV.13) is that it provides multiple prediction filters in order to respond effectively to fluctuations in the frequency distribution of the signal. This system employs near instantaneous compression to compress the dynamic range. The prediction filters and noise-shaping filters of the system switch simultaneously in order to minimize the energy of output noise.

Only first order and second-order digital filters are used in the encoder.

The encoder selects which predictor is most suitable in the manner defined below.

- The predictor adaptation unit in Figures IV.12 and IV.13 compares the peak value of the prediction errors sampled over 28 samples in a sound unit and then selects the filter which generates the minimum peak.
- Prediction errors which are chosen are gain-controlled (normalized by their maximum value) and noise shaping is executed at the same time.

As a result of the above mentioned strategy, first-order and second order filters are used for signals in the low and middle frequencies and the straight pass through filter is used for high frequency signals. This is done in order to obtain a high instantaneous Signal/Noise ratio.

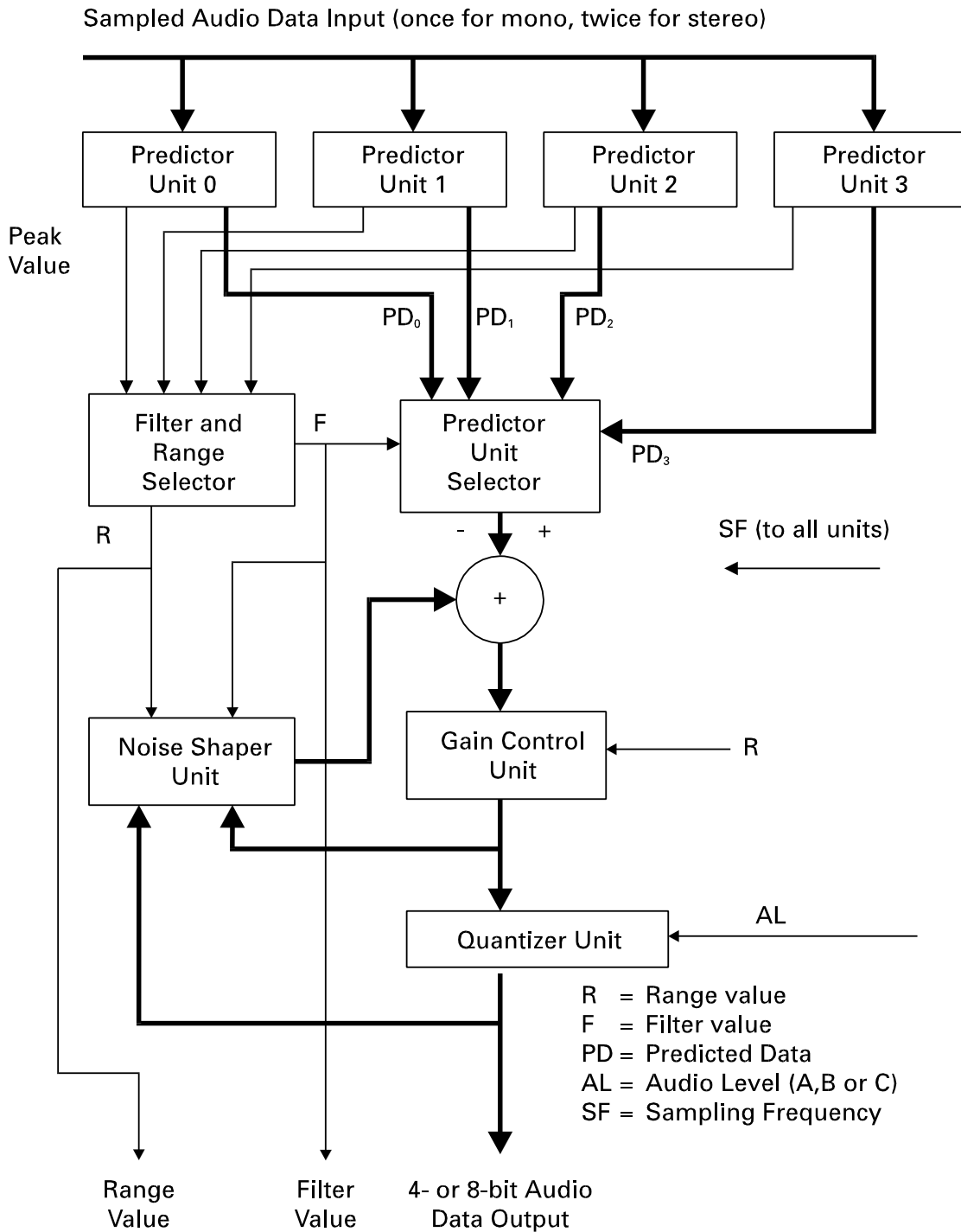
Different strategies of selecting filters are allowed. The above set is only one approach.

All data processing has to be done in 2's complement code with a sufficient number of bits to avoid extra quantization distortion.

The quantizer output of 4- or 8-bit audio data is a rounded value when inputted.

IV.4 Audio Data Encoding

Figure IV.13 General configuration of the ADPCM



## 4.2 ADPCM Encoder

Figure IV.13 gives the general configuration of the ADPCM encoder which needs to be done once for mono and twice for stereo.

If the input data has been recorded with emphasis (see the CD-DA Specification, page 1A) then the emphasis bit in the coding information byte has a value of 1; otherwise it has the value 0.

There are four predictor units indexed here as 0 to 3 (see Figure IV.13 and VI.14).

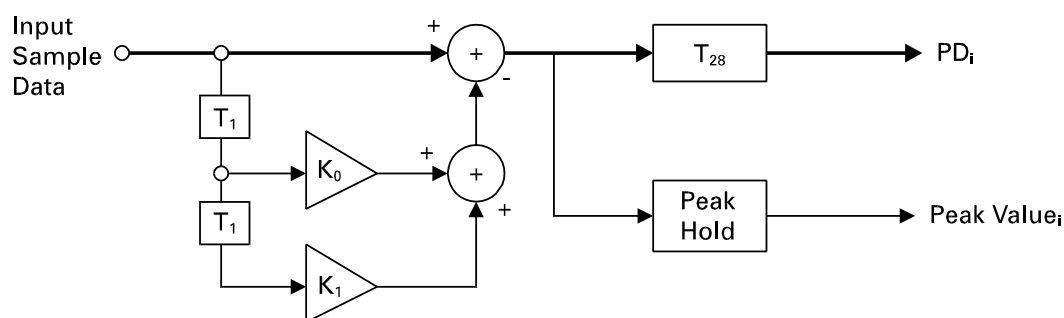
The filter and range selector generates the values called filter (F) and range (R) (see IV.4.3) depending on the peak values from the predictor units. The predictor unit with the lowest peak value is selected.

The predictor selector unit sends the chosen data to the gain control unit. The gain control unit multiplies the input data value with the gain factor  $2^R$  where R is the range value (see IV.4.3).

The purpose of the noise shaper unit (see Figure IV.15) is to compensate for extra noise introduced in the predictor unit.

The output value is rounded to 8 or 4 bits of audio data in the quantizer depending on the required audio level (A, B or C).

Figure IV.14 Predictor Unit



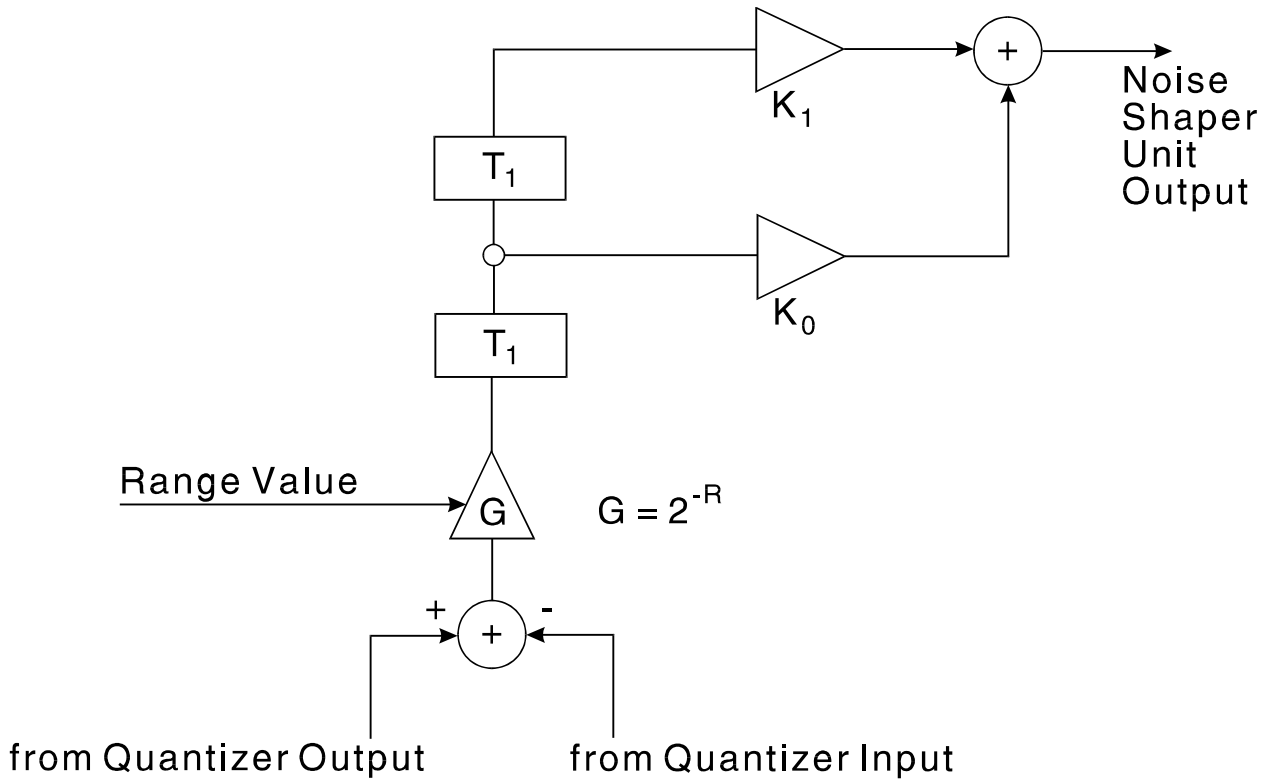
$T_1$  = one sample delay

$T_{28}$  = 28 sample delay

$K_0$  and  $K_1$  are gain factors and depend on the filter selected (see IV.4.3)

$i$  = 0 to 3

Figure IV.15 Noise Shaper Unit



$T_1$  = one sample delay

$K_0$  and  $K_1$  are gain factors and depend on the filter selected values.

**4.3 Sound Parameters**

Figure IV.16 shows the values of the gain for the various filters of the sound parameter and Figure IV.17 shows the range values.

Figure IV.16 **Gain values for sound parameter filters**

Filter	$K_0$	$K_1$
0	0 0.000000*	0 0.000000*
1	0.9375 0.111100*	0 0.000000*
2	1.796875 1.110011*	-0.8125 -0.110100*
3	1.53125 1.100010*	-0.859375 -0.110111*

(\*) Base 2 rational numbers

Figure IV.17 **Range values for sound quality levels**

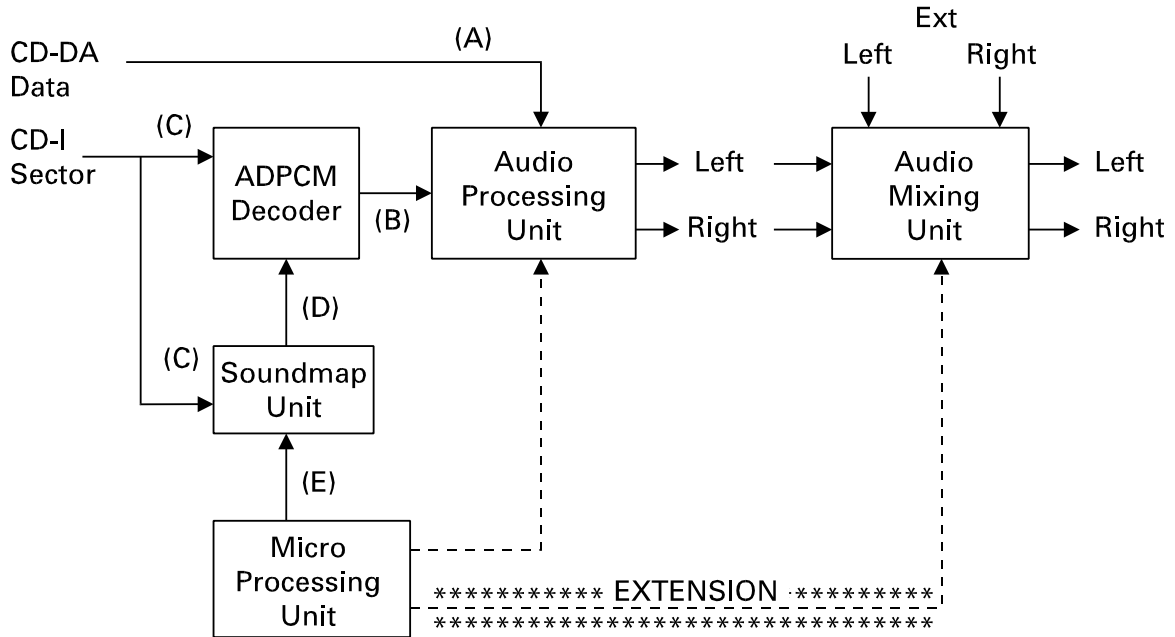
Sound Quality Levels			
	A	B	C
Range Values	0 to 8	0 to 12	0 to 12



IV.5 ADPCM Decoder

5.1 Audio Decoder Model

Figure IV.18 Audio Decoder Model



— = data flow  
 - - - = control

A = CD-DA data from CD-DA track,  
 B = decoded ADPCM audio data from CD-I sectors,  
 C = ADPCM audio data from CD-I Form 2 sectors to a soundmap or directly to the ADPCM decoder,  
 D = ADPCM audio data from a soundmap,  
 E = soundmap data from the microprocessing unit<sup>1</sup>  
 \*\*\*\*\* EXTENSION \*\*\*\*\*  
 Ext = external audio. EXTENSION  
 \*\*\*\*\*

<sup>1</sup> The microprocessing unit can generate soundmap data from an application program under CD-I or external control. The microprocessing unit may be based on a CPU or a dedicated coprocessor coupled to the CPU. This is an implementation issue.

## 5.2 Audio Processing Unit

The audio processing unit (see Figure IV.18) converts digitally coded audio information to the analog outputs called left and right.

This unit can process audio data coming from a CD-DA track on the disc (A) or audio data coming from the ADPCM decoder (B).

This unit also supports audio mixing and attenuation control (see IV.6.3).

\*\*\*\*\* EXTENSION \*\*\*\*\*

After the audio processing unit an audio mixing unit may be used to mix and/or attenuate external audio signals (e.g. synthesizer) with CD-I generated audio under the control of the microprocessing unit.

\*\*\*\*\*

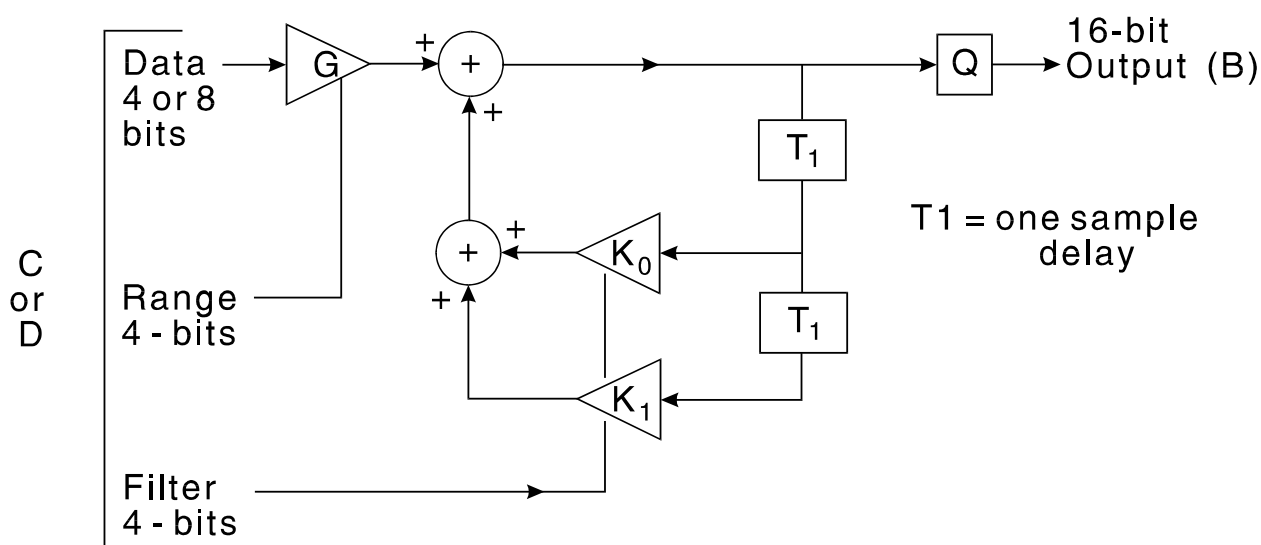
### 5.3 ADPCM Decoder

The ADPCM decoder (see Figure IV.19) converts the CD-I audio sector encoded data to 2's complement 16-bit PCM coded audio (B). The sampling frequency depends however on the audio quality level used (see IV.2).

This unit has to buffer the CD-I audio data in order to decode the time-multiplexed signals. The input can come from the Compact Disc Control Unit (see VIII.1) directly (C) or from a sound map in the CD-I system memory (D) (see Figure IV.18).

A 4 or 8-bit data word is converted into a 16-bit word by the ADPCM decoder (see Figure IV.19).

Figure IV.19 ADPCM Decoder



The multiplier G adjusts the gain of the system by means of the range value so that the audio data is decoded into 16-bit words.

IV.5 ADPCM Decoder

---

The multiplier algorithm is given below.

For level A the multiplier algorithm is:

$$\text{Word value} = \text{Audio data value} * 2^{(8-R)}$$

For levels B and C the multiplier algorithm is:

$$\text{Word value} = \text{Audio data value} * 2^{(12-R)}$$

In each case the Range (R) is:

$$\text{Range} = \text{Range data value (see Figure IV.17)}$$

The filter data controls the coefficients  $K_0$  and  $K_1$  of the filter. These coefficients are the same as those used for the encoder (see Figure IV.15)

The output data is quantized to 16 bits by the quantizer Q. If overflow occurs then these must be limited (clipped) to the maximum values.

### 5.4 Soundmap Unit

The soundmap unit holds one or more soundmaps created by the CDFM (see Figure IV.18). These are located in the CD-I system memory (RAM) and contain CD-I formatted audio data. The soundmap can be filled from the Compact Disc Control Unit (see C in Figure IV.18 or VIII.2) or from the microprocessing unit (E). The soundmap cannot be filled with CD-DA coded audio from the disc.

The CD-I audio sector data bytes going to the ADPCM decoder cannot be sent concurrently to the soundmap unit.

Normally the soundmap contains, in RAM, the whole or a part of the audio part of a real-time record. It is possible to fill one soundmap from disc (see Figure IV.18;C) or from the micro- processing unit (see Figure IV.18; E) and, in real time, send data to the ADPCM decoder (see Figure IV.18;D) from another soundmap concurrently.

It is recommended that the soundmap contains an integer number of contiguous CD-I audio sectors (each containing 18 sound groups) in RAM. The memory representation of a soundmap is that of a contiguous set of sound groups.

Note that the 20 last unused bytes of each CD-I audio sector are, consequently, not present in a soundmap.

### 5.5 Microprocessing Unit

The microprocessing unit (see Figure IV.18) can rearrange audio data in a soundmap. It can also generate new audio data into a soundmap under software control for special sound effects or phonetically coded speech.

This unit is responsible for controlling the mixing and attenuation of the:

- Audio Processing Unit.

\*\*\*\*\* EXTENSION \*\*\*\*\*

- Audio Mixing Unit.

\*\*\*\*\*

## IV.6 Audio Data Rearrangements

---

### IV.6 Audio Data Rearrangements

#### 6.1 Non Real-time Rearrangements

The application may create one or more soundmaps and fill them with data from the disc or from data within the application itself. The number of soundmaps and their size is limited only by the total amount of RAM.

If necessary, two soundmaps can be combined into a single soundmap for output to the audio processor. Two mono soundmaps may be combined into a stereo soundmap, or the left (or right) channel of one stereo soundmap may be combined with the left (or right) channel of a second stereo soundmap to create a third stereo soundmap.

The two soundmaps must always be of the same audio quality level.

When combining two large soundmaps, it is considered a non-real-time operation because the sound is not being output as this mixing is done.

IV.6 Audio Data Rearrangements

6.2 Real-time Rearrangements

In IV.5.4 it was stated that it is allowed to transfer CD-I audio data from disc to a soundmap while concurrently playing back audio from a second soundmap.

By using the mixing feature of CDFM (see VII.2.2.4.2, SD\_MMix and SD\_SMix), it is possible to read the data from the disc one sector at a time, mix it with a sector in memory and output the combined data to the audio processor with only a small delay in the audio signal. It is required that it takes less than 1/75 of a second (13 ms) to mix two one sector buffers.

This would allow an application to have a soundtrack on the disc and mix in sound effects from soundmaps in memory (see Figure IV.20a and 20b).

In this example, the application would set the play control lists to alternate between filling soundmap B and soundmap C. As each of these buffers is filled it is mixed with a sector from soundmap A into soundmap D or E respectively. Finally, soundmaps D or E are output to the audio processor.

In this manner, the actual delay in the soundtrack is less than 13 ms from its unmixed time. To avoid a delay when initiating the soundmap mixing it is recommended to mix the soundtrack with a sector of silence so that the delay is constant.

Figure IV.20a Soundmaps and Real Time Audio Rearrangements

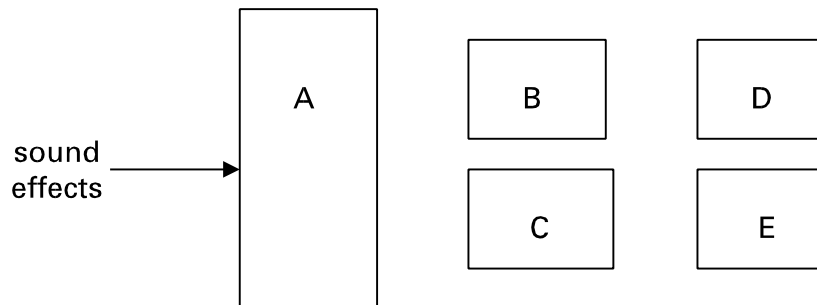
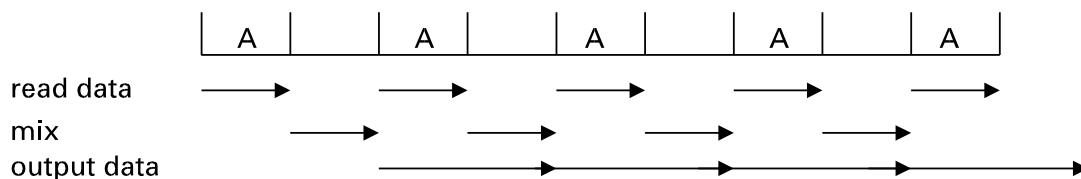


Figure IV.20b Real Time Audio Rearrangements Timing





IV.6 Audio Data Rearrangements

---

**6.3 Audio Mixing Control Unit**

The Audio mixing control unit, which is a part of the audio processing unit (see Figure IV.21), consists of four attenuators between left-in and left-out, left-in and right-out, right-in and right-out, and right-in and left-out. The attenuation is controlled by the CDFM in steps of 1 db.

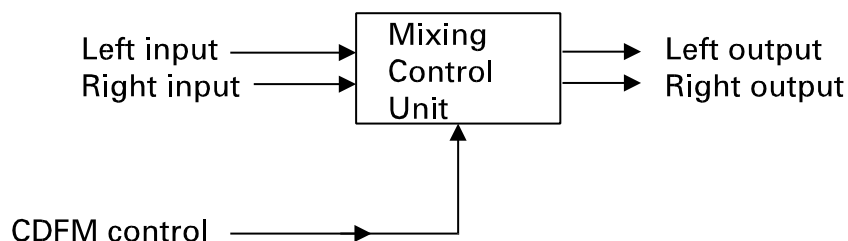
The control value for the attenuators is an 8-bit value.

If bit 7 is zero (range 0 to 127), then bit 6 to bit 0 defines the attenuation value in db.

If bit 7 is one then the signal at the output of the attenuator should be muted.

The Audio mixing control unit can control both the volume and balance for a stereo signal and the volume and panning for two mono signals<sup>2</sup>.

Figure IV.21 **Audio Mixing Control Unit**



\*\*\*\*\* EXTENSION \*\*\*\*\*

The mixing control in the audio mixing unit (see Figure IV.18) has the same capabilities as the audio mixing control unit of the audio processing unit defined above.

\*\*\*\*\*

---

<sup>2</sup> It is the responsibility of the application to define the attenuators control values so that an output level does not exceed any of the input levels.

IV.7 Extended Playing Time

---

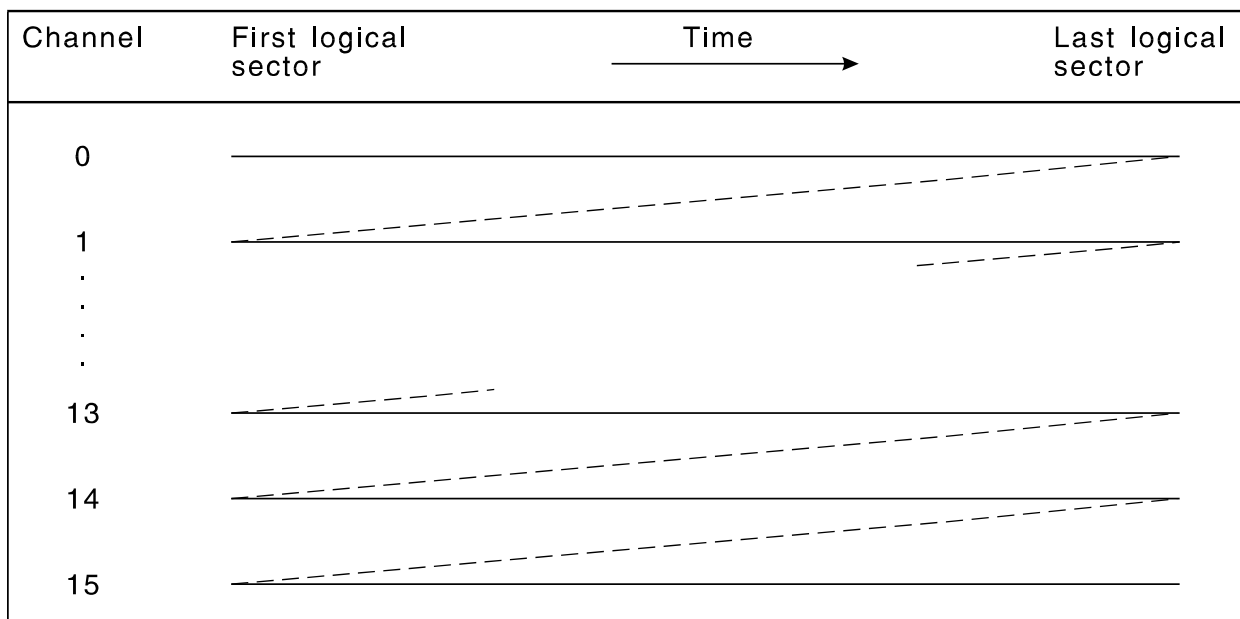
**IV.7 Extended Audio Playing Time**

Longer playing times can be achieved in two ways. The first is by playing the CD-I track more than once with a different channel number selected for each scan.

For level C audio, there is a maximum of 16 concurrent channels available.

Figure IV.22 is an example of longer playing time with this first approach using level C audio.

**Figure IV.22 Example of extended audio playing time using level C audio**



———— = Real-time playback  
 ---- = Access of first logical sector of next channel

### IV.7 Extended Playing Time

---

If the disc is replayed 16 times with 16 different channel numbers, then the total playing time relative to a CD-DA play time will increase sixteenfold. A delay of a few seconds may be experienced between the end of one channel and the beginning of a new channel, depending on the length of the CD-I track.

The second way of achieving extended playing time may be achieved by filling a soundmap with a 'non-real-time' audio record while another previously filled soundmap is used for playback in real-time. When the last soundmap is empty the next soundmap is used for playback and another soundmap can be filled from the disc. In the case of level C the filling rate is, at most, a factor of 16 times faster than the playback rate. In this example the player has to pause to keep up with the average data rate. This method gives, for level (C) mono a total playing time of 16 times that for CD-DA without any delays.

This page is intentionally left blank

Table of Contents

---

**Chapter V Real-time Video Data Representations**

	<b>Page</b>	
V.1	General	V-1
	1.1 Introduction	V-1
	1.2 Bit & Byte Ordering and Reserved Conventions	V-2
	1.3 The Video Encoding/Decoding Chain	V-3
V.2	Visual Presentation/Image Structure	V-4
	2.1 General	V-4
	2.2 Resolution	V-5
	2.2.1 Normal Resolution	V-5
	2.2.2 Derived Resolutions	V-6
	2.3 Safety Area	V-8
	2.4 Recommended Display Timing	V-9
	2.4.1 Pixel Aspect Ratios	V-10
	2.5 Image Sizes	V-11
	2.6 Planes	V-12
	2.7 Sub-screens	V-13
V.3	Image Encoding	V-14
	3.1 Image Formats	V-14
	3.1.1 525 and 625 Line Dedicated Formats	V-14
	3.1.2 525/625 Line Compatible Format	V-14
	3.1.3 Other Image Formats	V-15
	3.1.4 Meaning of 'Compatibility'	V-16
	3.2 Encoder Timing and Image Sizes	V-17
	3.3 Image Coding Methods	V-18
	3.4 Natural Images: DYUV	V-19
	3.4.1 DYUV Encoding Model	V-19
	3.4.1.1 Matrixing and Normalisation	V-20
	3.4.1.2 Filtering and Subsampling	V-22
	3.4.1.3 Delta Coding	V-23
	3.5 High Resolution Natural Images: DYUV + QHY	V-25
	3.5.1 The Coding Scheme	V-25
	3.5.2 Encoding the QHY Data	V-28
	3.5.3 QHY Quantization Levels	V-28

This page is intentionally left blank

## Table of Contents

	<b>Page</b>
3.6 Graphics and Text	V-30
3.6.1 General	V-30
3.6.2 Absolute RGB	V-30
3.6.3 Color Lookup Table	V-31
3.7 Runlength Coding	V-32
3.7.1 Run-coded 7-bit CLUT	V-32
3.7.2 Run-coded 3-bit CLUT	V-32
V.4 The Video Decoder	V-33
4.1 General Description	V-33
4.2 Display Scanning	V-38
4.3 Planes and Paths	V-39
4.4 Image Representation and Decoding	V-39
4.4.1 General	V-39
4.4.1.1 Representations in Memory	V-39
4.4.1.2 RGB Levels	V-40
4.4.2 DYUV	V-40
4.4.2.1 Representation in Memory	V-40
4.4.2.2 Decoding Model	V-41
4.4.2.3 DYUV Absolute Start Values	V-43
4.4.3 High Resolution DYUV + QHY	V-44
4.4.3.1 Representation in Memory	V-44
4.4.3.2 Decoding Model	V-44
4.4.3.3 Use of QHY for High Resolution Graphics	V-46
4.4.3.4 Loading QHY Quantization Levels	V-46
4.4.4 Absolute RGB	V-47
4.4.5 CLUT	V-48
4.4.5.1 8-bit CLUT	V-48
4.4.5.2 7-bit CLUT	V-48
4.4.5.3 4-bit CLUT	V-48

This page is intentionally left blank



## Table of Contents

	<b>Page</b>
4.4.6 Runlength Coded CLUT	V-49
4.4.6.1 Runlength 7-bit CLUT	V-49
4.4.6.2 Runlength 3-bit CLUT	V-50
4.4.7 CLUT Organization	V-51
4.4.8 Allowed Image Coding Combinations	V-52
4.4.9 Allowed Image Coding Resolutions	V-53
4.5 Display Control Facilities	V-54
4.5.1 Display Control Program	V-54
4.5.2 Line Pointer Tables	V-57
4.5.2.1 Image Line Pointer Table	V-57
4.5.2.2 Display Line Start Pointers	V-57
4.6 Display Control Functions	V-58
4.6.1 Selecting Image Coding Methods	V-58
4.6.2 Loading DYUV Start Values	V-62
4.7 Error Concealment	V-63
4.7.1 General	V-63
4.7.2 Even/Odd Line Separation	V-63
4.7.3 Concealment Techniques	V-64
4.8 525/625 Line Image Interchange	V-65
V.5 Visual Effects	V-66
5.1 General	V-66
5.2 Cut	V-66
5.3 Scroll	V-67
5.3.1 Image Positioning	V-67
5.3.2 Scroll	V-68
5.3.2.1 Scroll of DYUV Images	V-69
5.3.2.2 Scroll of Run-length Images	V-69
5.4 Partial Update	V-70
5.4.1 Rectangular Updates	V-70
5.4.2 Irregular Updates	V-70
5.4.3 Partial Update of DYUV Images	V-71
5.4.4 Partial Update of Runlength Images	V-71

This page is intentionally left blank

## Table of Contents

	<b>Page</b>
5.5 CLUT Update	V-72
5.5.1 CLUT Animation	V-72
5.5.2 Dynamic CLUT Update	V-73
5.5.3 Dual 7-bit CLUT	V-73
5.6 Synchronization to Display Scanning	V-74
5.7 Overlay	V-75
5.7.1 Plane Order	V-75
5.7.2 Transparency Mechanisms	V-76
5.7.2.1 Transparency Bit	V-76
5.7.2.2 Color Key	V-76
5.7.2.3 Transparency via Mattes	V-77
5.7.3 Transparency Control	V-78
5.8 Wipes	V-79
5.8.1 Two-plane Wipes	V-79
5.8.2 Single-plane Wipes	V-79
5.9 Image Contribution Factors	V-80
5.9.1 Image Mixing	V-81
5.9.2 Fades and Dissolves	V-81
5.10 Mattes	V-82
5.10.1 Matte Mechanism	V-82
5.10.2 Matte Commands	V-84
5.10.3 Loading the Matte Control Registers	V-85
5.11 Mosaics	V-86
5.11.1 Basic Facilities	V-86
5.11.1.1 Pixel Hold	V-87
5.11.1.2 Pixel Repeat	V-88
5.11.1.3 Line Hold/Repeat	V-88
5.11.2 Effects	V-89
5.11.2.1 Granulation	V-89
5.11.2.2 Magnification/Zoom	V-89
5.11.2.3 Reduced Resolution	V-90

This page is intentionally left blank

Table of Contents

---

	<b>Page</b>	
5.12	Cursor	V-91
5.13	Backdrop	V-92
5.14	Interlace	V-93
V.6	On-disc Coding Formats	V-94
6.1	General	V-94
6.2	Video Data Sequences	V-95
6.2.1	Packing of VDSQs	V-95
6.3	Video Coding Information Byte	V-96
6.3.1	Format	V-96
6.3.2	Example of Application Specific Coding	V-97
6.4	Image Formats	V-98
6.4.1	General	V-98
6.4.2	RGB555 Images	V-99
6.4.3	Error Concealment	V-100
6.4.4	High Resolution Images	V-101
6.4.4.1	DYUV & QHY	V-101
6.4.4.2	Compatibility with the Base Case	V-101
6.5	Pixel Data Formats	V-102
6.5.1	Natural Images	V-102
6.5.1.1	DYUV	V-102
6.5.1.2	QHY	V-103
6.5.2	RGB555	V-104
6.5.3	Color Lookup Table	V-105
6.5.3.1	8-bit CLUT	V-105
6.5.3.2	7-bit CLUT	V-105
6.5.3.3	4-bit CLUT	V-105
6.5.4	Run Length Encoded Images	V-106
6.5.4.1	Run-coded 7-bit CLUT	V-106
6.5.4.2	Run-coded 3-bit CLUT	V-107

This page is intentionally left blank

List of Illustrations

---

<b>Fig. no.</b>	<b>Caption</b>	<b>Page</b>
V.1	Example of Bit Ordering	V-2
V.2	Example of Multiple Byte Ordering	V-2
V.3	Video Encoding/Decoding Chain	V-3
V.4	Normal Full-screen Display Resolutions	V-5
V.5	Example of normal and double resolution	V-6
V.6	Horizontal/Vertical Multiplication Factors for Resolutions	V-7
V.7	Safety Area	V-8
V.8	Safety Area Dimensions in Normal Resolution	V-8
V.9	Recommended Horizontal Display Timings	V-9
V.10	Pixel Aspect Ratios	V-10
V.11	Displayed Image Formed from Superimposed Planes	V-12
V.12	Example of the Use of Subscreens	V-13
V.13	Aspect Ratio Distortions	V-15
V.14	Horizontal Resolutions and Timing for Image Encoding in Normal Resolution	V-17
V.15	Vertical Dimensions of Full and Safety Area Images	V-17
V.16	Encoding Scheme for DYUV Images	V-21
V.17	Sub-sampling Phase Relationships in the Horizontal Direction	V-22
V.18	Quantizing and Coding Table for Delta Coding	V-24
V.19	Coding Scheme for High Resolution DYUV + QHY	V-27
V.20	Two-dimensional Interpolation Filter Coefficients used in the Encoder/Decoder	V-28
V.21	Typical QHY Quantization Levels with Associated Code Values	V-29

This page is intentionally left blank



List of Illustrations

---

<b>Fig. no.</b>	<b>Caption</b>	<b>Page</b>
V.22	CLUT Sizes	V-31
V.23	Video Decoder Model	V-33
V.24	Real-Time Decoder	V-34
V.25	Overlay and Mixer	V-35
V.26	Image Store	V-36
V.27	Display Control using the Display Control Program	V-37
V.28	Possible Combinations of Image Planes	V-39
V.29	Pixel Pair Coding Scheme	V-40
V.30	DYUV Coded Image Decoding Process	V-41
V.31	QHY Pixel Pair Scheme	V-44
V.32	QHY High Resolution Mode Decoding Process	V-45
V.33	Load QHY Quantization Levels Instruction	V-46
V.34	Set QHY Bank Instruction	V-46
V.35	Memory Representation Absolute RGB	V-47
V.36	Memory Representation 8-bit CLUT	V-48
V.37	Memory Representation 7-bit CLUT	V-48
V.38	Memory Representation 4-bit CLUT	V-48
V.39	Memory Representation RL 7-bit CLUT	V-49
V.40	Memory Representation RL 3-bit CLUT	V-50
V.41	CLUT Organization	V-51
V.42	Allowed Image Coding Resolutions	V-53
V.43	Control Program Instructions available for Path 0 only	V-55
V.44	Control Program Instructions available for Path 1 only	V-55

This page is intentionally left blank

## CD-I Full Functional Specification

### Chapter V

### Video Real-time Data Representations

#### List of Illustrations

---

		<b>Page</b>
V.45	Control Program Instructions available for both Paths	V-56
V.46	Load Display Line Start Pointer Instruction	V-57
V.47	Select Image Coding Methods Instruction	V-59
V.48	Coding Values CM0 amd CM1	V-59
V.49	Load Display Parameters Instruction	V-60
V.50	Allowed Image Coding Method Combinations	V-61
V.51	Load DYUV Start Value Instruction	V-62
V.52	Odd/Even Line Separation in Memory	V-63
V.53	525/625 Line Image Interchange	V-65
V.54	Positioning of Displayed Image within Larger Image	V-67
V.55	Vertical Scroll Mechanism	V-68
V.56	Example Rectangular and Irregular Partial Updates	V-70
V.57	Partial Updates Transition Areas	V-71
V.58	Load CLUT Color Instruction	V-72
V.59	Set CLUT Bank Instruction	V-72
V.60	Interrupt at Scan Line Instruction	V-74
V.61	Load Plane Order Instruction	V-75
V.62	Load Transparent Color instruction	V-76
V.63	Load Mask Color Instruction	V-76
V.64	Load Transparency Control Instruction	V-78
V.65	Load Image Contribution Factor Instruction	V-80
V.66	Example of Overlapping and Non-overlapping Mattes	V-82

This page is intentionally left blank

## CD-I Full Functional Specification

### Chapter V

### Video Real-time Data Representations

#### List of Illustrations

---

		<b>Page</b>
V.67	Format Matte Command	V-84
V.68	Load Matte Control Register Instruction	V-85
V.69	Generating Mosaic Effects	V-86
V.70	Example of Pixel Hold Function	V-87
V.71	Load Mosaic Pixel Hold Factor Instruction	V-87
V.72	Example of Pixel Repeat Function	V-88
V.73	Example of Granulation	V-89
V.74	Example of Image Magnification Effect using Pixel and Line Repeat	V-90
V.75	Load Backdrop Color Instruction	V-92
V.76	Video Data sequence	V-95
V.77	Format Video Coding Information Byte	V-96
V.78	Video Data Element	V-97
V.79	Video Data Element after Expansion	V-97
V.80	Example of RGB555 Image Format	V-99
V.81	High Resolution Image on Disc	V-101
V.82	Video Data Element DYUV	V-102
V.83	Video Data Elements QHY	V-103
V.84	Video Data Elements RGB555	V-104
V.85	Video Data Element 8-bit CLUT	V-105
V.86	Video Data Element 7-bit CLUT	V-105
V.87	Video Data Element 4-bit CLUT	V-105
V.88	Video Data Elements RL7-bit CLUT	V-106
V.89	Video Data Elements RL3-bit CLUT	V-107

This page is intentionally left blank

## **Chapter V Video Real-time Data Representations**

### **V.1 General**

#### **I.1 Introduction**

This chapter specifies the CD-I video functions and data formats. The organisation of the chapter is as follows. The basic visual structure and properties of images are defined in section 2. Section 3 specifies how images are encoded, including encoding models for the various image coding methods, and provisions for compatibility between 525 line and 625 line display systems.

Section 4 specifies the functions of the video decoder, including the formats of images in memory, decoding models and display control functions (the Display Control Program, V.4.5). Section 5 specifies the CD-I visual effects, both the primary functions available via the Display Control Program and higher level functions built on these. Finally, the formats on disc of images and display control information are specified in section 6.

In this chapter, the terms "path 0" and "path 1" are used interchangeably with the terms "plane A" and "plane B".

V.1 General

---

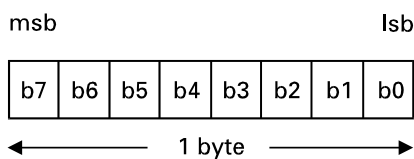
**1.2 Bit & Byte Ordering and Reserved Conventions**

**Bit Ordering**

In this specification, the graphical representation of all multiple-bit quantities is such that the most significant bit is on the left and the least significant bit is on the right.

Figure V.1 **Example of Bit Ordering**

Example for 8 bits:



where b = bit

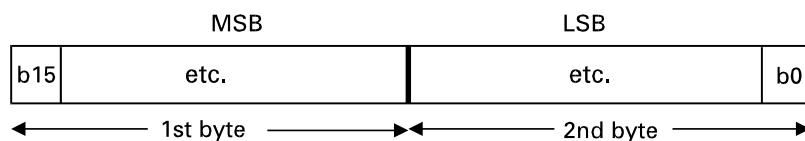
**Byte Ordering**

Quantities which require more than 8 bits for their representation are held in more than one byte on disc. For all such quantities, the ordering of bytes on disc (as seen at the interface to the disc driver) is such that the Most Significant Byte is first and the Least Significant Byte is last.

Multiple-byte quantities are represented graphically such that the left-hand-most byte is most significant and the right-hand-most byte is least significant.

Figure V.2 **Example of Multiple Byte Ordering**

Example - 16 bit quantity:



Where b = bit

Multiple-byte quantities in memory are represented such that the memory address of successive bytes increases from left to right.

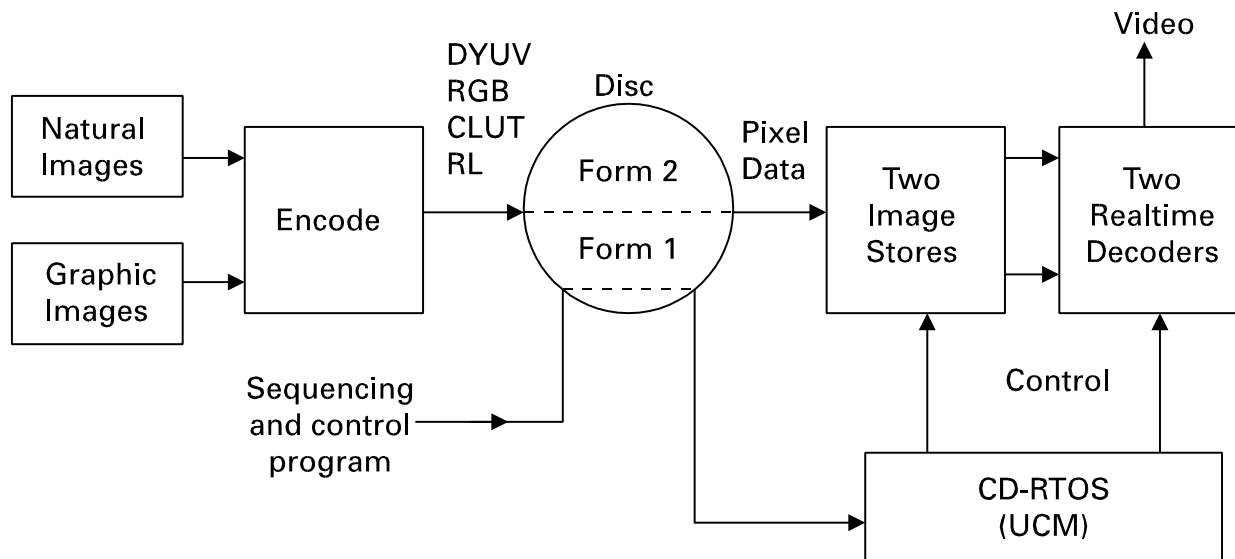
**Reserved**

It should be noted that **all bit fields that are reserved contain the symbol ♦**. These fields are set to zero in the present document.



### 1.3 The Video Encoding/Decoding Chain

Figure V.3 Video Encoding/Decoding Chain



The Video Encoding/Decoding Chain is shown in Figure V.3 above. Natural images and graphical images are encoded according to one of a number of specified methods (DYUV, RGB, Colour Lookup Table (CLUT) and Run-length (RL)), and the resultant pixel data stream is placed in Form 2 video sectors on the CD-I disc along with Form 1 data sectors containing program data for sequencing and visual effects, and image-related data.

In the player, the program in conjunction with the User Communication Manager (UCM, see VII.2.3), loads the pixel data into an image store, and controls its subsequent display via a real-time video decoder. There are two image stores, each with its own decoder, to allow two independent image planes.

UCM is specified in VII.2.3. The other parts of the encoder/ decoder chain are specified in this chapter.

V.2 Visual Presentation/Image Structure

---

**V.2 Visual Presentation /Image Structure**

**2.1 General**

This section describes the structure of an image as seen by the viewer.

CD-I images are intended to be displayable on standard 525 line (NTSC) or 625 line (PAL/SECAM) televisions or monitors.

For 525 line systems, slightly different parameters are specified for monitors and TV sets. This is in order to allow an optimized display for monitors, while guaranteeing no loss of critical information at the edges of the screen on a TV set.

No matter how the image is stored on the disc, it is always displayable via all decoders, whether 525 or 625 line.

## V.2 Visual Presentation/Image Structure

---

### 2.2 Resolution

The displayed image is a rectangular array of pixels, whose dimensions, timing and resolution are optimized to the target 525 and 625 line display systems taking into account the following requirements:

- the Base Case System (see Chapter VIII) has a display screen of normal 'consumer TV' resolution.
- the aspect ratio of pixels should be close to unity, for use with graphical applications
- a specified 'safety area' of the screen should have guaranteed visibility under all circumstances
- it should be possible to display 525/625 line compatible images with minimum distortion (<10%)

#### 2.2.1 Normal Resolution

The Normal Resolution full-screen display dimensions for 525 and 625 line systems are shown in Figure V.4. We will refer to these dimensions as the horizontal and vertical display resolutions.

Figure V.4 **Normal Full-screen Display Resolutions**

Display	Number of pixels horizontally	Number of pixels vertically
525 line monitors	360	240
525 line TVs	384	240
625 line systems	384	280

The horizontal resolution for 525 line monitors (360) is less than that for 625 line systems (384) in order to improve the 525 line pixel aspect ratio (see Figure V.10) and to reduce aspect ratio distortion when displaying 525/625 compatible images (Figure V.13).

Also, the horizontal resolution for 525 line TVs (384) is greater than that for 525 line monitors in order to ensure that the safety area is always fully displayed under worst-case overscan conditions.

V.2 Visual Presentation/Image Structure

---

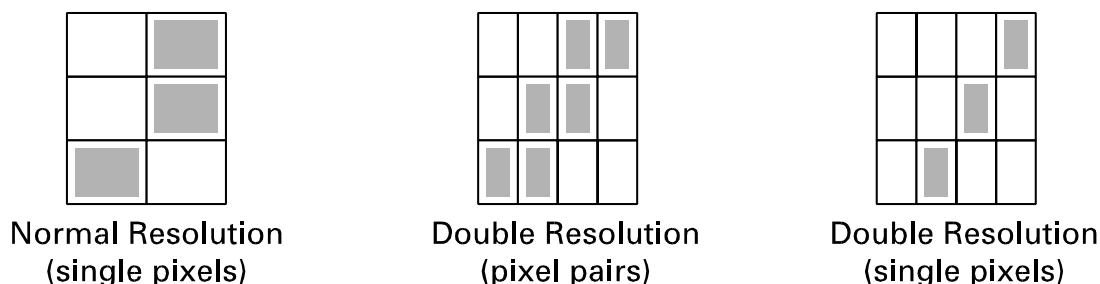
**2.2.2 Derived Resolutions**

Double Resolution is defined with twice the horizontal resolution, and the same vertical resolution, as Normal Resolution. However, the resolution of a Base Case display screen is not sufficient to display a single pixel with full quality in Double Resolution. Therefore, Double Resolution is aimed at improved display of graphical images in the following two cases:

- where Double Resolution horizontal pixel pairs can give increased positional accuracy compared with single Normal Resolution pixels.
- where a single pixel's reduced display quality in Double Resolution does not reduce the identifiability of the graphic object or character (e.g. Kanji) to an unacceptable level.

The display of normal and double resolution is illustrated by the following examples:

**Figure V.5 Example of Normal and Double Resolution**



\*\*\*\*\* EXTENSION \*\*\*\*\*

High Resolution is defined with twice the resolution both horizontally and vertically as Normal Resolution. It is intended for use with high resolution display screens so is available (with one exception, see V.4.4.9) as an extension only. Where high resolution is available in the Base Case, it must be used with restrictions as specified above for double resolution, but applied in the vertical as well as the horizontal direction.

\*\*\*\*\*

These formats are derived from the Normal Resolution by multiplication by the factors in the Figure V.3.

Figure V.6

**Horizontal/Vertical Multiplication Factors for Resolutions**

Resolution	Horizontal Factor	Vertical Factor
Normal	1	1
Double	2	1
High	2	2

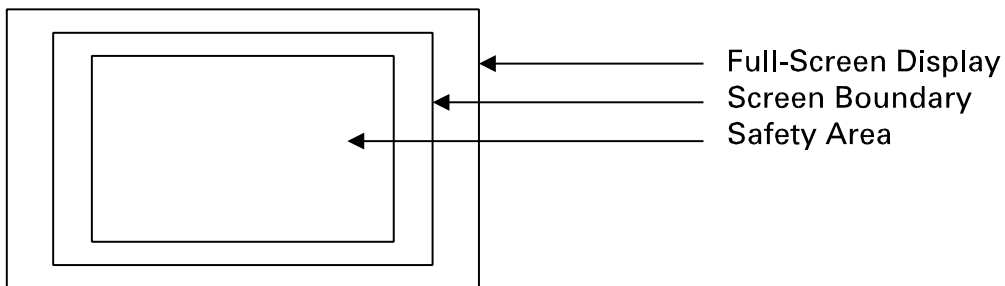
## Examples:

- Double Resolution for 525 line monitors = 720 x 240 pixels
- High Resolution for 625 line systems = 768 x 560 pixels

### 2.3 Safety Area

In order to prevent the loss of vital information (e.g. text) the concept of a 'Safety Area' is introduced. The Safety Area, shown in Figure V.7, is defined as a reduced area on the screen surface, where display of information is guaranteed despite all tolerance values that can occur in monitors and TV sets.

Figure V.7 Safety Area



The dimensions of the Display Safety Area, in Normal Resolution, are shown in Figure V.8.

Figure V.8 Safety Area Dimensions in Normal Resolution

Display	Number of pixels horizontally	Number of pixels vertically
525 line systems	320	210
625 line systems	320	250

These figures must be multiplied by the factors in Figure V.6 for other resolutions. For example, for 625 line systems and High Resolution, the Safety Area is 640 x 500 pixels.

## 2.4 Recommended Display Timing

The full active line scan period (52.4  $\mu$ s) is **recommended** for use in 525 line systems. For 625 line systems, it is **recommended** that the horizontal display period is compressed to 50  $\mu$ s. This improves the aspect ratio distortion of 525/625 line compatible images, while still ensuring that the whole screen is filled under worst case display system scanning tolerances.

Thus the **recommended** horizontal timings are shown in Figure V.9.

Figure V.9 **Recommended Horizontal Display Timings**

Display	Time in $\mu$ s		$f_{\text{pix}}$ (MHz)
	Full Screen	Safety Area	
525 line monitors	52.4	46.6	6.87
525 line TVs	52.4	43.7	7.32
625 line systems	50	41.7	7.68

$f_{\text{pix}}$  = Pixel clock frequency

V.2 Visual Presentation/Image Structure

---

**2.4.1 Pixel Aspect Ratios**

For display of computer graphic images, a close approach to square pixels is desirable. The pixel aspect ratios for 525 and 625 line systems, using the recommended horizontal timings in Figure V.9, are computed as follows

Aspect Ratio = Pixel Height / Pixel Width

$$= \frac{\text{pixel clock frequency} * 52.4 \mu\text{s}}{\text{Number of active lines}} * \frac{3}{4}$$

where the number of active lines = 242.5 (525 lines)

= 287.5 (625 lines)

The pixel aspect ratios are shown in Figure V.10.

**Figure V.10 Pixel Aspect Ratios**

Display	Aspect Ratio
525 line monitors	1.11
525 line TVs	1.19
625 line systems	1.05



## 2.5 Image Sizes

Images may be encoded and displayed which are larger than full-screen, either horizontally, vertically or both. In this case only a part of the image is displayed at any time. This part may be positioned anywhere within the image (with some restrictions for some coding methods, see V.5.3)

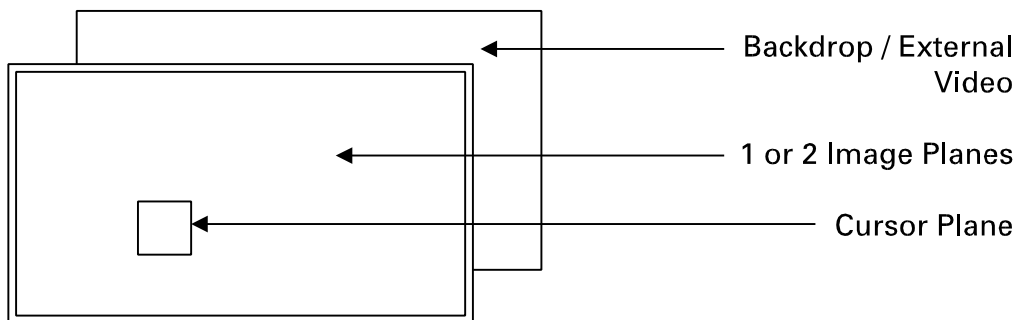
Images with sizes smaller than full-screen may also be encoded. However, if the width of an image is smaller than the full-screen width, it may not be displayed directly, but only as a partial update to an image which has at least full-screen width (see V.5.4). If the height of an image is less than full-screen height, but it has width greater than or equal to full-screen width, it may be displayed as a subscreen (See V.2.7).

## 2.6 Planes

The displayed image may be formed by the superimposition of a number of component images, which are considered to be displayed on up to 4 separate planes placed one behind the other as shown in Figure V.11.

One or two image planes are defined, depending on the coding scheme used for the images (see V.4.3).

Figure V.11 **Displayed Image Formed from Superimposed Planes**



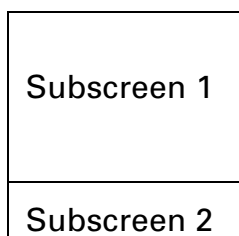
The Backdrop plane may be external video (extended case only), or a constant color. This color must be the same over the whole of any horizontal line, but may be different for different lines.

The cursor plane has dimensions less than those of the image planes, but may be positioned at any coordinate over the other planes. (If a larger cursor is required, it can be generated by software techniques in an image plane).

## 2.7 Subscreens

An image plane may be split up into an arbitrary number of horizontal strips or subscreens. Subscreens may be any height from one line to the height of the screen. Different planes may be split into differently sized subscreens. An example of Subscreens is shown in Figure V.12.

Figure V.12 **Example of the Use of Subscreens**



Each subscreen may contain an image of different coding type and/or resolution.

Example: Subscreen 1 - Normal Resolution natural image  
Subscreen 2 - Double Resolution text/graphics

### V.3 Image Encoding

---

#### V.3 Image Encoding

This section specifies the way that images are encoded, i.e. the image coding techniques and algorithms that are used. For the details of the coding formats on disc, see V.6.

##### 3.1 Image Formats

###### 3.1.1 525 and 625 Line Dedicated Formats

Two different image encoding formats are specified, differing in size. These are:

1. a 525 line dedicated format comprising 360 x 240 pixels in normal resolution and corresponding to the 525 line monitor format.
2. a 625 line dedicated format comprising 384 x 280 pixels in normal resolution and corresponding to the 625 line format.

These formats are intended for use where distortion in aspect ratio, or direct coupling to other equipment using 525 or 625 line formats are important factors. They have zero aspect ratio distortion on 525 line monitors and 625 line systems respectively. For local markets, this type of format gives simplicity in the authoring stage.

###### 3.1.2 525/625 Line Compatible Format

For non-local and international markets, a minimal distortion 'Compatible' format is specified. Discs with images coded to this format can be distributed worldwide with only small distortions in aspect ratio ( +/-3% for 525 line monitors and 625 line systems). Experiments have shown that distortions of up to +/-10% are typically not noticed when no direct comparison to the original can be made.

The Compatible format has dimensions the same as for 625 line format, but a horizontal 'stretch' or pre-distortion of the image is applied to equalize the aspect ratio distortions for 525 line monitors and 625 line displays. To achieve this, the encoder horizontal timing is specified such that the pixel aspect ratio is the geometric mean of that for 525 line monitors and 625 line systems, (Figure V.10).

Compatible aspect ratio =  $\sqrt{(1.05 * 1.11)} = 1.08$

This gives an aspect ratio distortion of +/-3.0%.

Figure V.13 shows the aspect ratio distortion for each format, when displayed on the three target systems.

Figure V.13 **Aspect Ratio Distortions**

Format	Display of Target System		
	525 line monitors	525 line TVs	625 line systems
525 line	0	+7	-6
625 line	+6	+13	0
Compatible	+3.0	+10	-3.0

For a definition of the term 'Compatible', see V 3.1.4.

### 3.1.3 Other Image Formats

The image formats specified above are supported by all decoders, with minimum application involvement. However, images may also be encoded to 525 line TV format, i.e. 384 x 240. See V.4.8 and Appendix V.2.

The aspect ratio pre-distortion employed for the Compatible format may also be applied to the 525 line format to improve the aspect ratio distortion when viewed on a 625 line decoder.

In this case, the pre-distortion required is a 3% horizontal 'squeeze' rather than a 'stretch', corresponding to a 3% decrease in effective encoding pixel clock frequency. The aspect ratio distortions of this format on the three target systems are the same as for the Compatible format.

### 3.1.4 Meaning of 'Compatibility'

It is mandatory that all CD-I discs 'work' on all CD-I decoders. By this is meant:

- all essential information is presented to the user
- the user has access to all functions of the application on disc.

This capability is the responsibility of the application, with some help from the decoder hardware and system software (see e.g. Appendix V.2). The main implication of this for images is that their safety areas must be fully visible on all decoders. In this sense, all images however encoded, are 'compatible'.

However, this specification reserves the term 'Compatible' to apply to the 'Compatible format' specified in V.3.1.2. This format has the additional virtues of both minimum aspect ratio distortion on all decoders, and of being 'full screen height' on all decoders.

## V.3 Image Encoding

**3.2 Encoder Timing and Image Sizes**

The horizontal dimensions and timing for encoding of the three main image formats in Normal Resolution are as shown in Figure V.14.

Figure V.14 **Horizontal Resolutions and Timing for Image Encoding in Normal Resolution**

Format	Full Screen		Safety Area		$f_{enc}$ (MHz)
	No. of Pixels	Time in $\mu s$	No. of Pixels	Time in $\mu s$	
525 line	360	52.4	320	46.6	6.87
625 line	384	50.0	320	41.7	7.68
Compatible	384	48.6	320	40.5	7.90

$f_{enc}$  = pixel encoding clock frequency.

The pixel numbers must be multiplied by the factors in Figure V.6 for other resolutions. These encoder timings may represent actual encoder clock frequencies or effective clock frequencies achieved by digital filtering and subsampling.

The vertical dimensions on disc of full-screen images and their safety areas for the three formats, are shown in Figure V.15.

Figure V.15 **Vertical Dimensions of Full and Safety Area Images**

Format	Full Screen	Safety Area
525 line	240	210
625 line	280	210
Compatible	280	210

Note that the height of the safety area of all three encoding formats is 210 lines, to ensure visibility in the case of display on a 525 line decoder.

All images are coded on disc to their individual dimensions (e.g. a 525 line full image is coded as 360 x 240 pixel codes on disc; see V.6.4.1).

### 3.3 Image Coding Methods

A number of different image coding methods are specified, each optimized to a different image type (natural image, computer graphics, text etc.)

The image coding and compression schemes take into account what is possible in consumer products, e.g. the decoders do not require excessive computing power. For ease of implementation, binary values are used for the width of the necessary data streams.

**Natural (photographic) images** may be encoded by a differential method (DYUV) which takes advantage of the statistical properties of natural pictures, plus the properties of the human eye, to achieve compression by a factor of 3. An extension of this coding scheme (DYUV + QHY) is available for high resolution natural images in the Extended Case; this achieves compression of typically 9 - 10.

**Computer Graphics and bit-mapped text** may be encoded as absolute RGB components (RGB555) or by use of a restricted number of colors contained in a Color Lookup Table (CLUT)

**Cartoon Style Images** may be compressed using one-dimensional runlength (RL) coding. The high degree of compression achieved, typically 8 - 12 greater than CLUT coding alone, allows full screen animation of suitable material.

For the allowed resolutions of each coding method see V.4.4.9.



### V.3 Image Encoding

---

#### 3.4 Natural Images: DYUV

Digitized natural images typically have a high correlation between adjacent pixel values. Encoding the differences between successive pixels, instead of coding the pixel values themselves, results in a significant improvement in efficiency. This is known as delta coding.

A further improvement can be obtained by exploiting the fact that the human eye is less sensitive to spatial colour variations than to intensity variations. This makes it possible to code the chrominance components (U & V) of an image with less bandwidth than the luminance component (Y).

The combined coding scheme is called **DELTA-YUV** or **DYUV**. The principal parameters selected are:

- One dimensional prediction.
- The bandwidth for U and V is 0.5 the bandwidth for Y.
- A non-uniform 16 level fixed quantizer is used.

##### 3.4.1 DYUV Encoding Model

The coding scheme is shown in Figure V.16. A general description is given below.

RGB signals from a suitable analogue source are digitized at a sampling rate in accordance with one of the standards given in V.3.2, to give high resolution eight bit pcm RGB source images. The RGB images are matrixed to give YUV images which are then sub-sampled vertically and horizontally.

Prior to sub-sampling it is necessary to filter the images with an appropriate two dimensional anti-aliasing filter. The sub-sampled Y, U and V images are separately coded as four bit delta images which are subsequently combined for storage on the disk.

The final encoded image has 8 bits/pixel, or 16 bits per pixel pair, which is the basic indivisible encoding unit. For **disc and decoder formats** see V.6.5.1 and V.4.4.2 respectively.

V.3 Image Encoding

---

**3.4.1.1 Matrixing and Normalisation**

Matrixing of the RGB signals to YUV follows standard video practice. The defining equations are:

$$\begin{aligned} Y &= 0.299 * R + 0.587 * G + 0.114 * B \\ U &= 0.564 * ( B - Y ) \\ V &= 0.713 * ( R - Y ) \end{aligned}$$

The normalizing factors are chosen such that if the range of the RGB signals is 0...1 then the range of the Y signal is 0...1 and the range of the U and V signals is -0.5...+0.5

In order to allow a safety margin to guard against overshoot and undershoot resulting from filtering, the ranges of the  $Y_c$ ,  $U_c$  and  $V_c$  binary components, which are coded as eight bit values, are restricted in accordance with CCIR recommendation 601.1 such that the range of the  $Y_c$  signal is 16 (black) to 235 (peak white). The range of the  $U_c$  and  $V_c$  signals is from 16 to 240 with an offset of 128. These values are summarised in Appendix V.1.

The equations for the coded and normalized values of Y, U and V are given by:

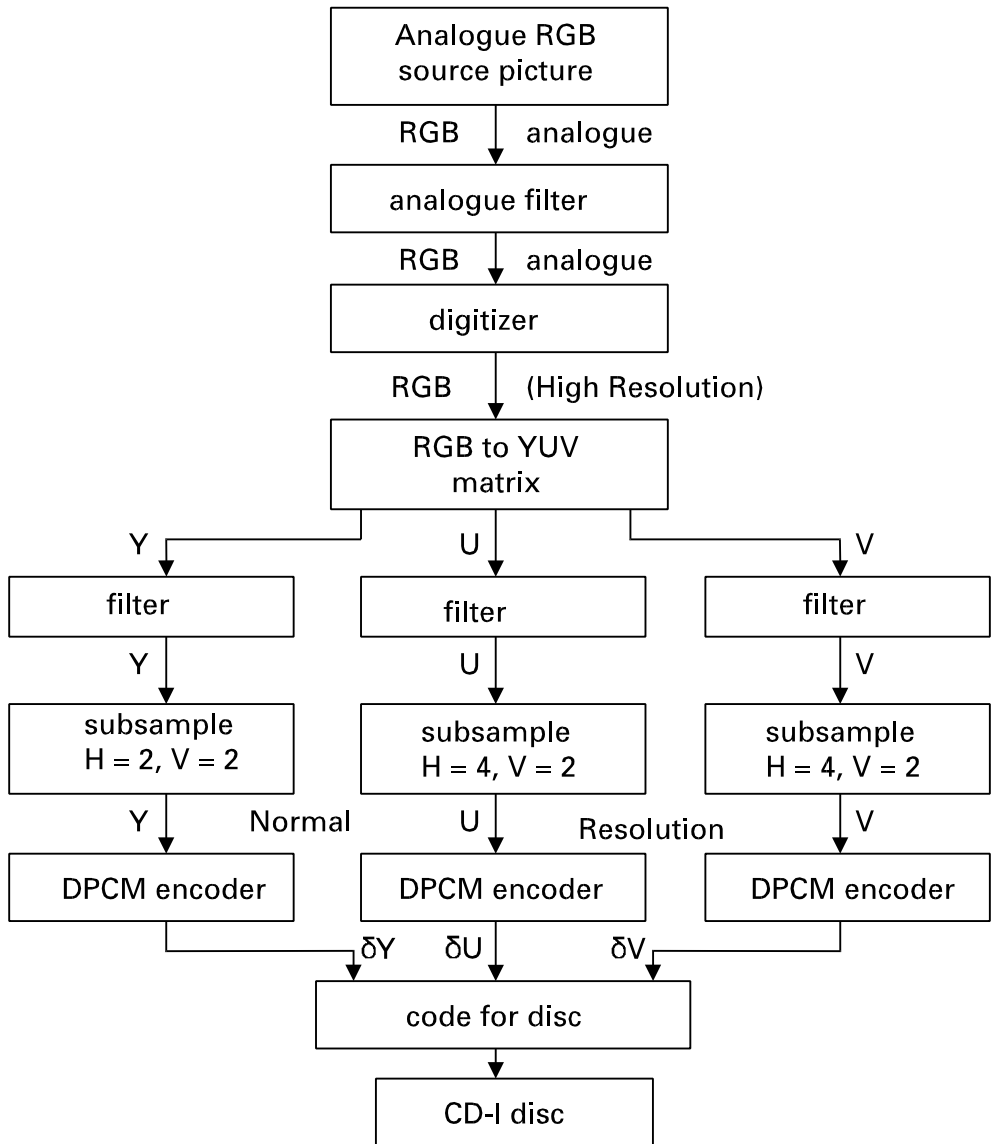
$$\begin{aligned} Y_c &= 219 * Y + 16 \\ U_c &= 224 * U + 128 \\ V_c &= 224 * V + 128 \end{aligned}$$

The matrixing and normalization of the YUV values is thus completely defined by the following encoding equations :

$$\begin{aligned} Y_c &= 65.5 * R + 128.5 * G + 25.0 * B + 16 \\ U_c &= -37.8 * R - 74.2 * G + 112.0 * B + 128 \\ V_c &= 112.0 * R - 93.8 * G - 18.2 * B + 128 \end{aligned}$$

Figure V.16 shows the encoding scheme for DYUV images

Figure V.16 **Encoding Scheme for DYUV Images**



DPMC: Delta Pulse Code Modulation

<sup>1</sup> H and V are Horizontal and Vertical subsampling factors

V.3 Image Encoding

---

3.4.1.2 Filtering and Sub-sampling

It is assumed that source images have been digitized at High Resolution. In order to code images at Normal Resolution it is necessary to sub-sample the source image. Sub-sampling is by a factor of two in the vertical direction both for the Y image and the UV images. The Y image is sub-sampled by a factor 2 horizontally and the UV images are sub-sampled horizontally by a factor 4.

The sub-sampling phase relationships in the horizontal direction are shown in Figure V.17.

Figure V.17

**Sub-sampling Phase Relationships in the Horizontal Direction**

High Res pixel nr.	0	1	2	3	4	5	6	7	8	9	10	11	12..
Normal Res pixel nr.	0	1		2		3		4		5		6..	
Luminance	Y	Y		Y		Y		Y		Y		Y..	
Chrominance	UV	UV				UV				UV..			
Pixel pair		----	0	----		----	1	----		----	2	----	...

nr. = number

Vertical subsampling is the same for Y, U and V components and is on even line numbers i.e. line numbers 0, 2, 4, ... etc.

In order to reduce the effects of signal aliasing it is necessary to filter the Y, U and V images prior to sub-sampling. It is **recommended** that digital filtering by means of FIR (Finite Impulse Response) filters is used, or some equivalent method that does not adversely affect the relative phases of the images. The CCIR signal levels (see Appendix V.1) allow some degree of overshoot or undershoot, resulting from filtering, but it is the responsibility of the filtering apparatus to constrain the magnitude of the filtered and subsampled images within the bounds 0...255.

**3.4.1.3 Delta Coding**

The components Y, U and V are separately coded using identical delta coding algorithms. Coding is by delta pcm using a one dimensional horizontal scheme.

For each component, the 4 bit transmitted code  $c_n$  for the  $n$ th pixel is the encoded quantized difference between the true value  $V_n$  for the  $n$ th pixel and the prediction value  $P_n$  for the  $n$ th pixel. The quantizing and coding table is defined in Figure V.18. Let  $Qf()$  denote quantization of difference values and coding according to that table and  $Qf^{-1}()$  denote decoding of code values to quantized difference values according to this table.

$$1) c_n = Qf( V_n - P_n )$$

The prediction value  $P_n$  is equal to the decoded value  $F_{n-1}$  for the previous pixel.

$$2) P_n = F_{n-1}$$

The decoded value  $F_n$  is calculated as the sum of the prediction value  $P_n$  and the decoded quantized difference value. The sum is performed modulo 256.

$$\begin{aligned} 3) F_n &= P_n + Qf^{-1}( c_n ) \\ &= F_{n-1} + Qf^{-1}( c_n ) \end{aligned}$$

The initial prediction value of  $P$  for each line is defined independently and transmitted separately as an absolute 8 bit value  $P_{abs}$

$$4) P_0 = P_{abs}$$

Data compression is achieved by using a non-uniform quantizer. A 16 level quantizer is used and so 4 bits per sample are transmitted. The scheme is non-adaptive and thus the quantizer is fixed as shown below. The same quantizer is used for luminance and both chrominance signals.

Because of the modulo 256 arithmetic that is employed, each output value is capable of representing a positive or, by means of wraparound, a negative quantized difference.

Figure V.18 **Quantizing and Coding Table for Delta Coding**

Input Difference		Code	Output Difference
0		0	0
1..... 2	- (254...255)	1	1
3..... 6	- (250...253)	2	4
7..... 12	- (244...249)	3	9
13.... 21	- (235...243)	4	16
22.... 35	- (221...234)	5	27
36.... 61	- (195...220)	6	44
62.... 99	- (157...194)	7	79
100...156	- (100...156)	8	128
156...194	- ( 62... 99)	9	177
195...220	- ( 36... 61)	10	212
221...234	- ( 22... 35)	11	229
235...243	- ( 13... 21)	12	240
244...249	- ( 7... 12)	13	247
250...253	- ( 3... 6)	14	252
254...255	- ( 1... 2)	15	255

**Overflow:** It is essential that the process of quantization does not result in inadvertent overflow or underflow (wraparound) of the decoded output as a result of errors intrinsic in the quantization process. The coding apparatus must be responsible for checking as the coding progresses that the true sum of the prediction value plus the quantized difference value is not outside the limits 0..255. In the event of these limits being exceeded, an adjacent quantized difference value must be substituted which will not cause overflow or underflow.

### V.3 Image Encoding

---

\*\*\*\*\* EXTENSION \*\*\*\*\*

#### 3.5 High Resolution Natural Images DYUV+QHY

In the Extended Case, High Resolution natural images may be encoded.

The DYUV method described in V.3.4.1 may be used for High Resolution natural images by doubling the horizontal and vertical sampling frequencies. This method gives the highest quality images. However the size of the encoded image is four times that for Normal Resolution. For compatibility, the image must, typically, be accompanied by a Normal Resolution image (see V.6.4.4). In this case the total size on disc and the loading time are both five times that for Normal Resolution.

The Quantized High Y (QHY) coding scheme described here allows these storage and loading times to be substantially decreased, while providing subjective quality close to that of full High Resolution DYUV.

##### 3.5.1 The Coding Scheme

The elements of the method are as follows (see Figure V.19).

The source image is digitized at High Resolution, matrixed to YUV, and filtered horizontally and vertically to give a Normal Resolution image, which is DYUV coded as described in V.3.4.1. The sub-sampling phases are as described there.

The Y component of the DYUV coded Normal Resolution image is decoded by the standard method and expanded by means of a 2 dimensional interpolation filter. This gives an image having the same number of picture elements as a High Resolution image but only the spatial resolution of the Normal Resolution image.

The interpolated Normal Resolution Y image is subtracted pixel by pixel from the original High Resolution Y image to give a Y difference image, representing both the high frequency component of the luminance and the DPCM quantization errors. This difference image tends to have large values only near sharp edges in the source image; its favourable statistics enable it to be very efficiently coded.

#### V.3 Image Encoding

---

The difference values are quantized and the resulting quantized values are runlength coded to give an auxiliary set of image data, the QHY (Quantized High Y) data.

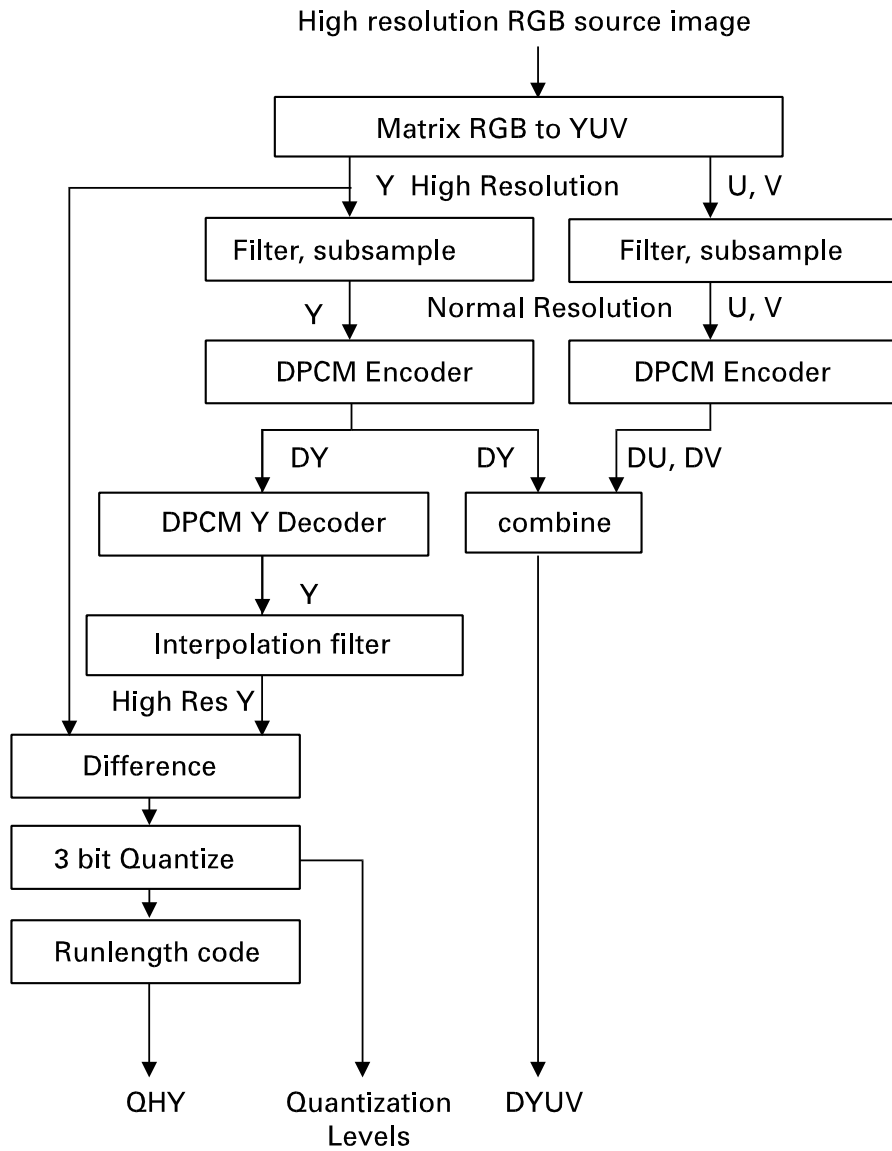
To display a High Resolution DYUV+QHY image the DYUV Normal Resolution image is decoded and interpolated to give a filtered Y image identical with that derived by the encoder. The runlength coded QHY image is decoded to give a High Resolution Y difference image. Adding the difference image to the filtered image gives rise to a restored High Resolution Y image that has good visual fidelity to the original source Y image.

Apart from interpolation, it is not necessary to process the U and V data, as the normal resolution U and V images are already of sufficiently high bandwidth.



V.3 Image Encoding

Figure V.19 **Coding Scheme for High Resolution DYUV + QHY**



V.3 Image Encoding

---

**3.5.2 Encoding the QHY Data**

The two-dimensional interpolation filter used in the encoder and decoder has the coefficients shown below in Figure V.20. This filter performs simple linear interpolation both horizontally and vertically. It is separable in the two dimensions and is designed to be simply implementable.

Figure V.20 **Two-dimensional Interpolation Filter Coefficients used in the Encoder/Decoder**

0.25	0.5	0.25
0.5	1.0	0.5
0.25	0.5	0.25

The filtered High Resolution image is constructed by inserting zeros between the Normal Resolution pixel values both vertically and horizontally, then by filtering with a 2-dimensional FIR (Finite Impulse Response) filter with coefficients shown in Figure V.20.

The Y difference values are coded with a 3-bit (8 level) quantizer (defined below) to give the QHY codes. These codes are then runlength coded as pairs of three-bit values to give the QHY data. The runlength coding format is the same as for run-coded 3-bit CLUT (RL3) images and makes use of the first 8 entries of CLUT bank 2 (see V.4.4.7). Disc and decoder formats are described in V.6.5.1.2 and V.4.4.3 respectively.

**3.5.3 QHY Quantization Levels**

The quantization levels are transmitted to the decoder along with the image (see Figure V.32). Therefore they may be chosen so as to best match particular images, or parts of images. Alternatively, a fixed set of levels may be used which is optimized for a range of source material. The choice of the threshold range for the zero quantization level has a particularly important effect on the degree of data compression that may be achieved. With typical source material and threshold levels, runlength compression results in an order of magnitude compression of the QHY data.

A typical set of quantization levels with their associated code values are shown in Figure V.21. In this example, QHY code 7 is used for run-coded text that is added to the high resolution image in areas of uniform brightness.

Figure V.21<sup>2</sup>

**Typical QHY Quantization Levels with Associated Code Values**

Difference value	Quantization levels Q	Coded value QL	QHY Code
-6 .... 6	0	128	0
7 .... 12	8	132	1
13 .... 20	16	136	2
21 .... 255	24	140	3
-7 .... -12	-8	124	4
-13 .... -20	-16	120	5
-21 ....-255	-24	116	6
Text	140	198	7

\*\*\*\*\*

---

<sup>2</sup> These levels are shown for the purpose of illustration, and may be varied to optimise the encoding of individual images or parts of images.

Quantization levels may take any even value in the range -256 ...254. Each level is coded as a single byte in offset binary form. If the level has a value Q, the coded value QL is given by

$$QL = Q / 2 + 128$$

The coded values QL are coded on disc in the same way as that used for CLUT values. For disc and decoder formats see V.6.5.1.2 and V.4.4.3.4 respectively.

### 3.6 Graphics and Text

#### 3.6.1 General

There are two fundamentally different methods of creating graphics or text on the screen.

- Downloading from disc the complete resultant image. This is described in this chapter and is mainly used for real-time or complex graphic images.
- Creating from a more compact representation which can be interpreted by system software. This is the preferred method for text and is used for simple graphic images. The methods specified for CD-I are given in VI.3.1 (text) and VII.2.3.2.9 (UCM).

For the direct downloading of graphic images, both direct absolute RGB coding and indirect CLUT (Color Lookup Table) coding are available.

#### 3.6.2 Absolute RGB

Images can be coded in R, G, B to an accuracy of 5 bits for each component, plus a remaining bit reserved for overlay information.

For each color component, the encoding law is:

$$C = (219 * C_A + 16)/8, \text{ rounded to the nearest integer}$$

where

- C = A color component (one of R,G,B)
- C<sub>A</sub> = The corresponding analog component, in the range 0 to 1
- C = 2 therefore represents the black level.

For disc and decoder formats see V.6.5.2 and V.4.4.4 respectively.

### 3.6.3 Color Lookup Table

This technique allows selection of a given number of colors from a much larger palette. The set of colors to be used for the image is loaded into a Color Lookup Table (CLUT) in the decoder.

Each pixel is then coded with the position in the CLUT of the color for that pixel. The colors in the CLUT may be chosen from a palette of  $2^{24}$  colors, i.e. R, G and B are each defined at 8-bit accuracy.

The CD-I system uses the following CLUT sizes

Figure V.22 **CLUT Sizes**

Mechanism	Colors	Available path	Purpose
8-bit	256 colors	path 0 only	CLUT8
7-bit	128 colors	path 0 and/or path 1	CLUT7, RL7
Dual 7-bit	128 x 2 colors	path 0 only	CLUT7, RL7
4-bit	16 colors	path 0 and/or path 1	CLUT4
3-bit	8 colors	path 0 and/or path 1	RL3

The available number of colors may be expanded by dynamic loading of new colors on a line by line basis.

For disc and decoder formats see V.6.5.3 and V.4.4.5 respectively.

### 3.7 Runlength Coding

Images may be runlength coded, with either 7 bits or 3 bits of color. The high degree of data compression may be used for animation of cartoon style images.

#### 3.7.1 Run-coded 7-bit CLUT (RL7)

Single pixels are coded as single bytes, giving the CLUT code for that pixel. A run of pixels of the same color is coded as two bytes, the CLUT code and the length of the run (from 2 to 255). A length of zero is used to indicate that the run extends until the end of the current line. All lines must be terminated with this code as an end marker. The coding is strictly on a line basis, so that there is no wraparound to the next line.

This scheme ensures that the compression ratio is never less than 1. It is typically of the order of 10 for cartoon style images.

RL7 may be used with a single 7-bit CLUT or with dual 7-bit CLUTs (see V.5.5.3).

Dynamic CLUT updates (see V.5.5.2) may be used to increase the number of colors in a run-coded image.

#### 3.7.2 Run-coded 3-bit CLUT (RL3)

The same coding scheme is used as for 7 bit runlength coding, except that the pixels are coded in pairs, one byte holding two 3-bit CLUT codes. The runlength specifies the number of times the pixel pair is repeated.

For disc and decoder formats see V.6.5.4 and V.4.4.6 respectively.

## V.4 The Video Decoder

### 4.1 General Description

The following functional block diagrams show the operation of the video decoder. Figure V.23 shows the high-level functions, and the Figures V.24 to V.27 show the operation of each block in more detail.

The diagrams are intended as a decoder model. They indicate the functions performed, and may, but need not, represent the actual hardware implementation in any particular decoder.

Figure V.23 Video Decoder Model

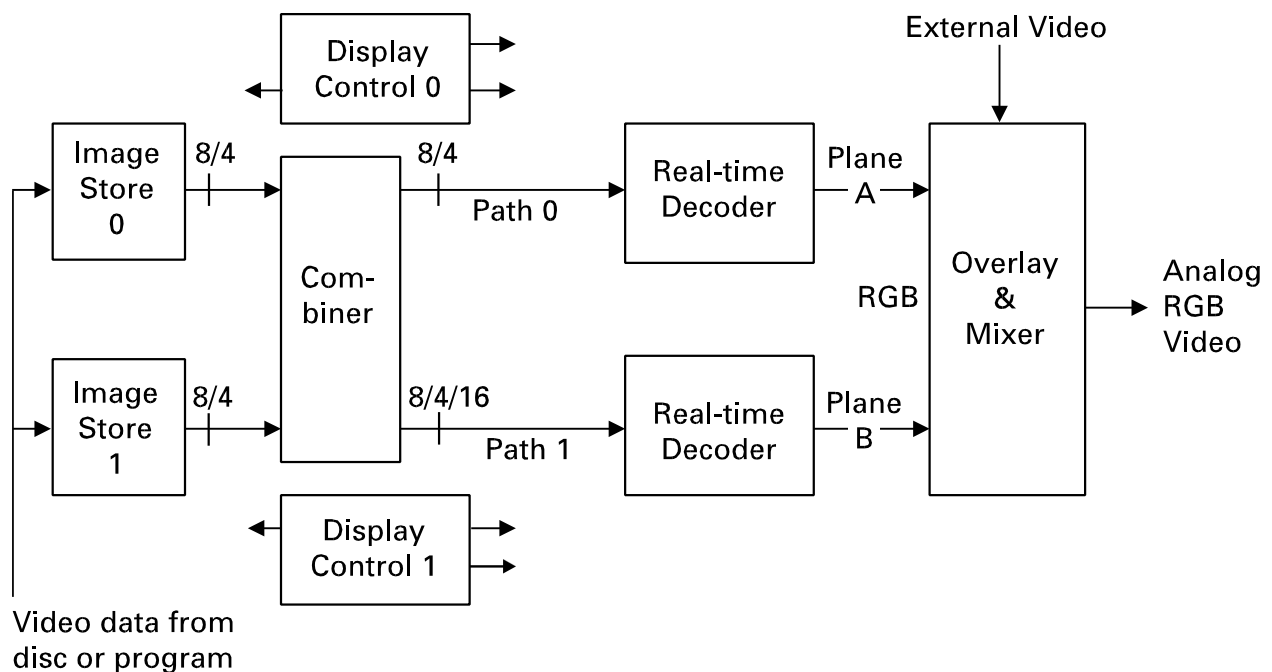


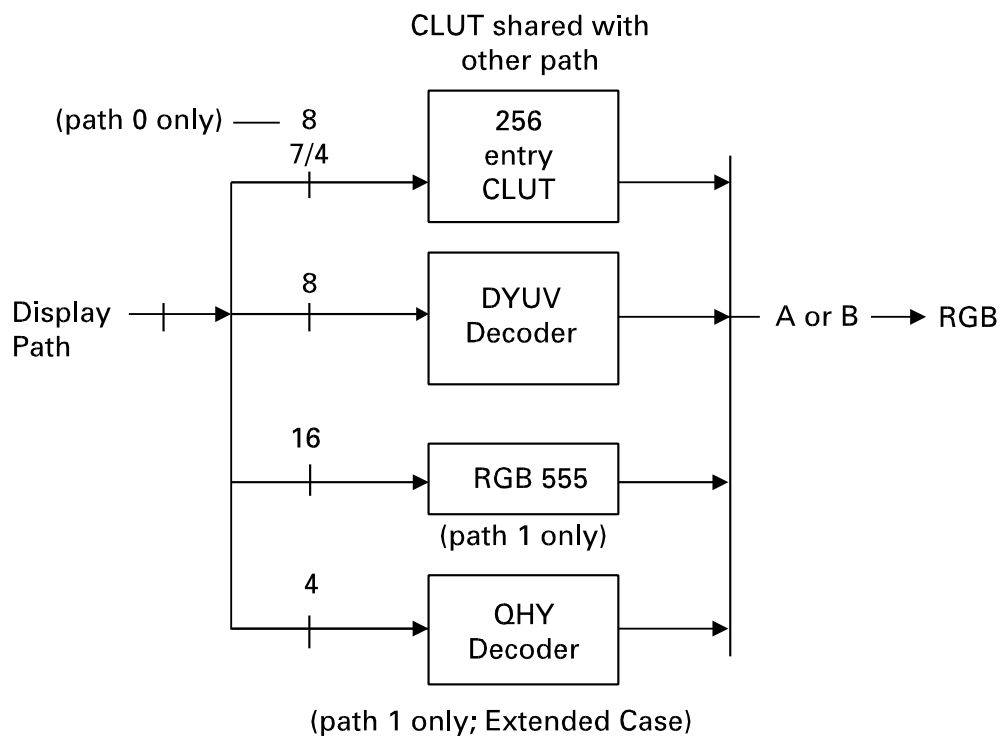
Image data for display is held in two image stores. Video data is loaded into one or both stores from disc or program. Each image store provides a real-time data stream at display scan rate from the image data that it holds. This data stream is 8 or 4 bits wide, depending on the resolution required. In the Base Case, only 4 bits are available at Double Resolution, and 8 at Normal Resolution.

V.4 The Video Decoder

The decoder contains two display paths, and the two data streams are normally used independently by these paths. However two 8-bit data streams may be combined to form one 16-bit stream in path 1.

Display paths 0 & 1 are decoded to RGB image planes A & B by two real-time decoders one of which is shown in Figure V.24.

Figure V.24 **Real-Time Decoder**

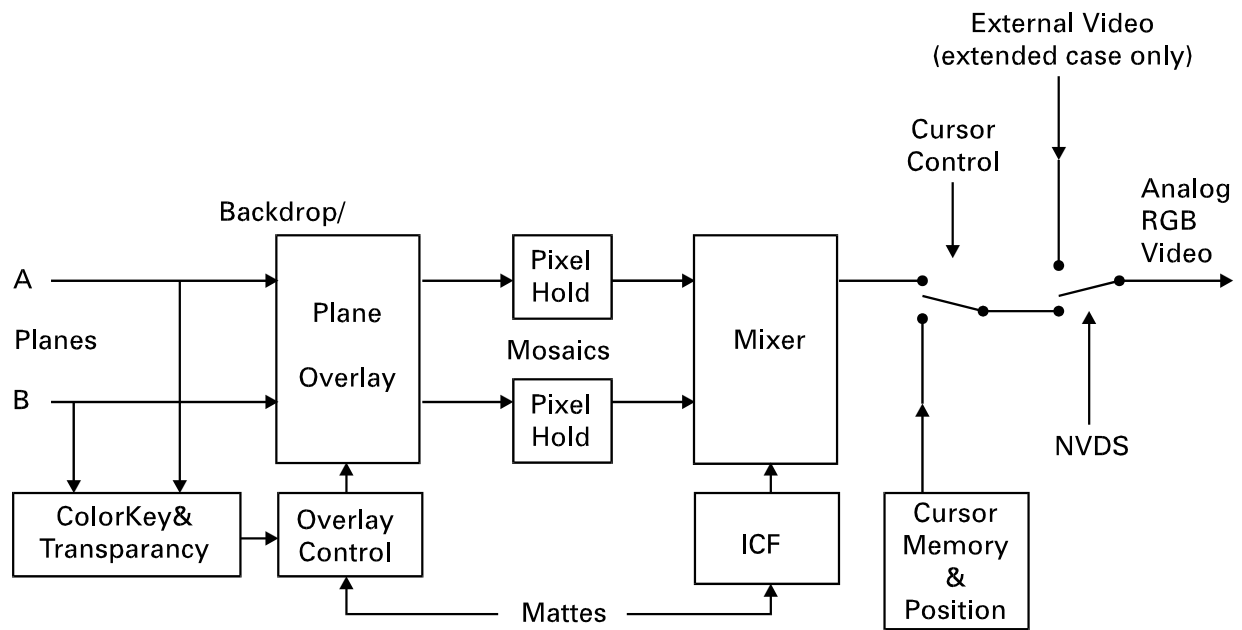


The image planes may be overlaid in any order on the screen (Figure V.25). Portions of the planes may be made transparent, in order to see the plane(s) behind them; this may be done by several mechanisms including color keying. Alternatively the planes may be mixed, that is their contents summed.

The contribution factor of each plane to the final displayed image is adjustable via an Image Contribution Factor. This may be used with either overlay or mixing. Finally, the cursor is overlaid on top of the image, and the backdrop (or, as an extension, external video) is displayed wherever both image planes are transparent.



Figure V.25 **Overlay and Mixer**



ICF = Image Contribution Factors  
 NVDS = Negated Video Data Select

V.4 The Video Decoder

---

Display control is performed by means of a display control program associated with each path, which includes display line start pointers (see V.4.5).

A number of visual effects facilities are provided, such as mosaics and mattes. These are specified in V.5.

Figure V.26 **Image Store**

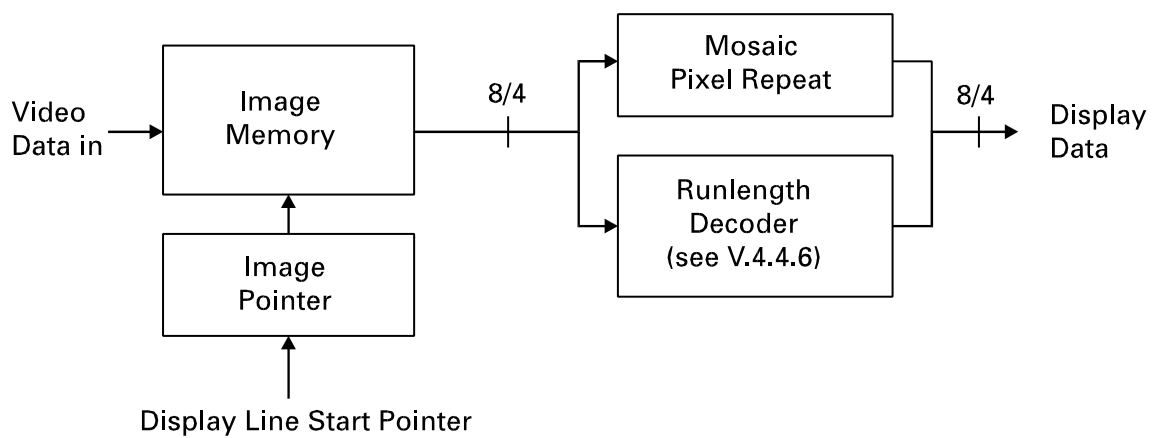
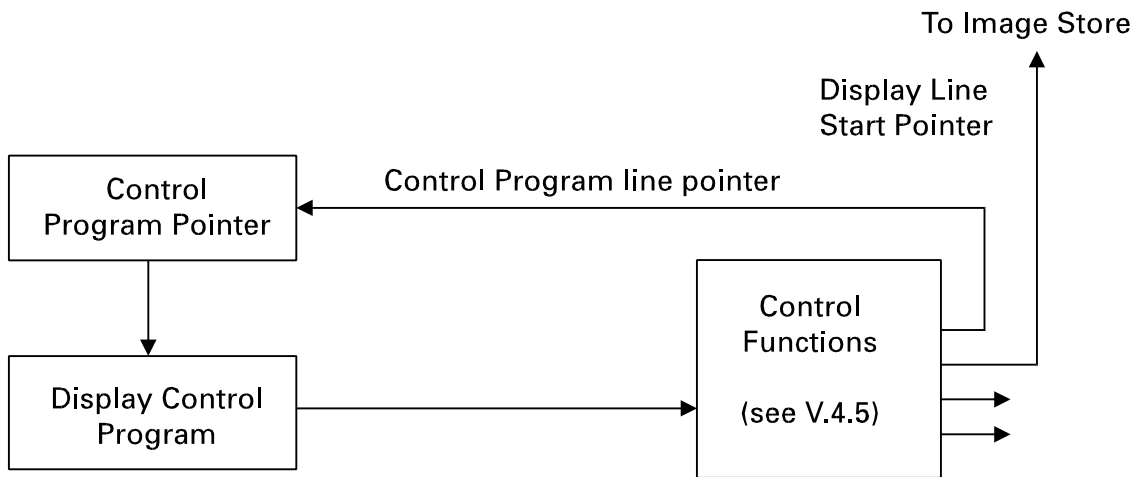


Figure V.27 **Display Control using the Display Control Program**



### 4.2 Display Scanning

The video decoder scan timing is such as to drive a standard 525 line (typically 60Hz) or 625 line (typically 50Hz) television display, with two fields per frame. For overlay over external video, the display must be capable of interlace.

If any other form of display system is used, it must simulate this timing from the point of view of the decoder.

\*\*\*\*\* EXTENSION \*\*\*\*\*

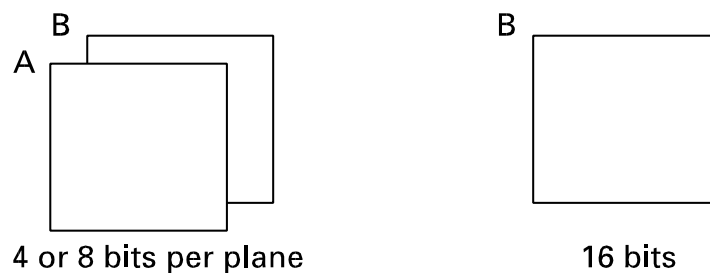
In the extended case, High Resolution images may be displayed with two interlaced fields per frame, or line sequentially. Note, however, that the order of execution of commands in the Display Control Program (DCP) (see V.4.6) will be different in these two cases. Applications should take this into account, and construct DCPs whose effects are independent of this order.

\*\*\*\*\*

### 4.3 Planes and Paths

The two path organization and the allowed data stream combinations described in V.4.1 imply the following possible combinations of image planes as shown in Figure V.28.

Figure V.28 **Possible Combinations of Image Planes**



Although the A plane is shown 'in front of' the B plane, the order of the planes may be set arbitrarily.

The cursor and backdrop planes are of course also present, as shown in Figure V.11.

## 4.4 Image Representation and Decoding

### 4.4.1 General

#### 4.4.1.1 Representations in Memory

Each horizontal line of pixels is represented in memory by a contiguous sequence of pixel codes at increasing memory addresses from left to right. However, consecutive lines of an image are **not** necessarily contiguous in memory.

In the case of RGB555 images which require combined data streams from the two image stores, each pixel code is split into two halves, each of which is arranged in such a contiguous sequence.

Notation: In this section, pixels are numbered consecutively from left to right as displayed. For bit and byte ordering conventions see V.1.2.

**4.4.1.2 RGB Levels**

All images, encoded by whatever method, are decoded (in terms of this model) to a uniform 8-bit linear representation of the Red, Green and Blue color components. For each component, black level is at 16 and nominal peak (white) level is at 235.

The analog output is given by:

$$C_A = (C-16)/219$$

where  $C_A$  = An analog component (one of R,G,B)  
 $C$  = The corresponding digital component.

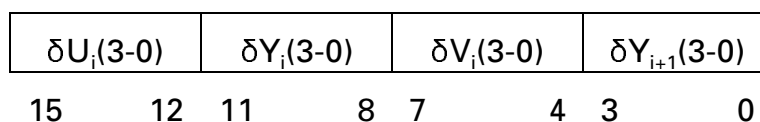
As the range of C is from 0-255, the range of  $C_A$  is -0.073 to 1.091.

**4.4.2 DYUV**

**4.4.2.1 Representation in Memory**

Each pixel pair (pixels i & i+1) is coded as two bytes as follows:

Figure V.29 Pixel Pair Coding Scheme



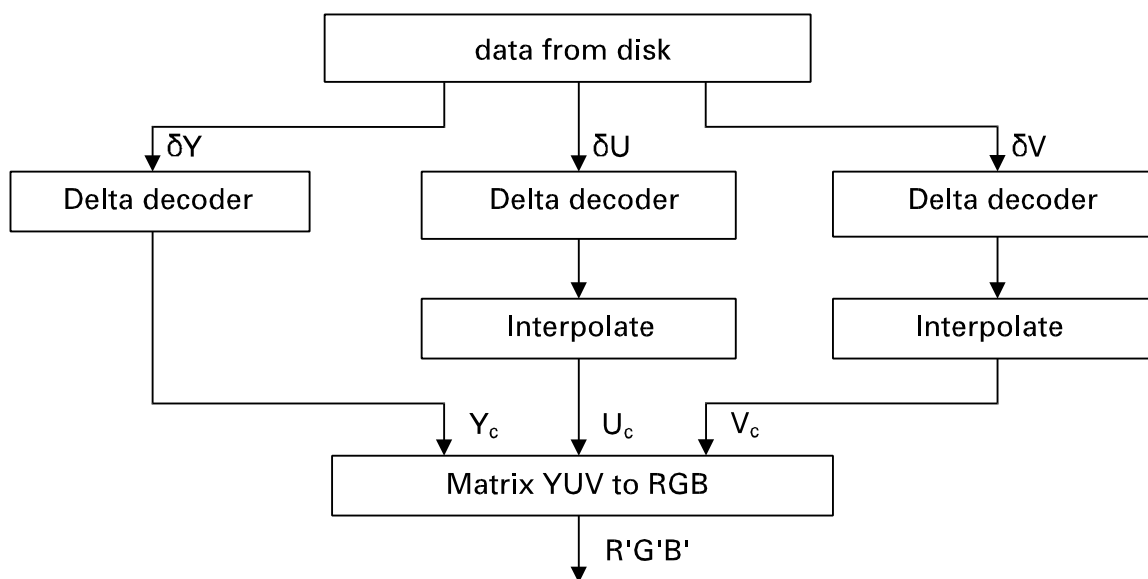
where i must be even. The pixel pair starts at an even X coordinate and at an even memory address.

$\delta Y$  = 4 bit luminance difference code.  
 $\delta U, \delta V$  = 4 bit chrominance difference codes.

## 4.4.2.2 Decoding Model

The process of decoding DYUV coded images is as shown in Figure V.30. The composite data in memory is separated into its delta pcm coded Y, U and V components. Each of the three components is decoded by a DPCM decoder to give an eight bit pcm image. Due to the subsampling the U and V images are only half resolution horizontally. It is thus necessary to interpolate the missing values. The normal resolution Y image  $Y_c$  is matrixed with the interpolated  $U_c$  and  $V_c$  images to give the eight bit digital RGB signals  $R'G'B'$ .

Figure V.30 DYUV Coded Image Decoding Process



**Delta decoding**

For each component Y, U and V, the same decoding scheme is used. For each picture element along a horizontal line the decoded value  $F_n$  is obtained from the coded value  $c_n$  by adding the decoded difference value  $Qf^{-1}(c_n)$  to the prediction value  $P_n$ . The prediction value  $P_n$  is equal to the previous decoded value  $F_{n-1}$ , except for the first element of the line, where the prediction value is set equal to the separately defined absolute value  $P_{abs}$ .

$$F_n = F_{n-1} + Qf^{-1}(c_n) \text{ if } n > 0$$

$$F_0 = P_{abs} + Qf^{-1}(c_0)$$

The notation used above is the same as that defined in V.3.4.1.

The same quantization table (Figure V.18) is used for defining the decoding function  $Qf^{-1}()$  as is used for encoding.

Addition is by modulo 256 arithmetic.

**Interpolation of U and V**

The values of U and V obtained from the delta decoder are at only half the horizontal resolution of the Y values, and so must be interpolated. Care must be taken to ensure that the inter-polation filter, which may perform simple linear interpolation, preserves the encoding phase relationships of V.3.4.1.2.



**Matrixing of YUV to RGB**

The Y value and the interpolated U and V values are matrixed to give RGB values. The matrix equations are given below.

If the decoded values of YUV are  $Y_c$   $U_c$   $V_c$  the values must first be de-normalised.

$$U' = (U_c - 128) * 1.733$$

$$V' = (V_c - 128) * 1.371$$

then

$$B' = Y_c + (U_c - 128) * 1.733$$

$$R' = Y_c + (V_c - 128) * 1.371$$

$$G' = (Y_c - 0.299 * R' - 0.114 * B') / 0.587$$

**NOTE:**

The decoded values R'G'B' will have nominal black levels of 16. Total excursions are in the range 0...255.

**4.4.2.3 DYUV Absolute Start Values**

The initial values of Y, U & V for each line ( $P_{abs}$  in the above equation for delta coding), are transmitted separately to the decoder and loaded via the display control program (V.4.5.1).

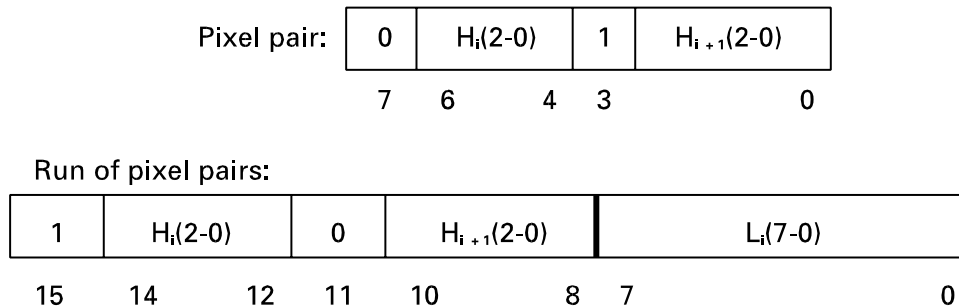
\*\*\*\*\* EXTENSION \*\*\*\*\*

### 4.4.3 High Resolution DYUV+QHY

#### 4.4.3.1 Representation in Memory

The DYUV component is held in memory in the Normal Resolution format. The QHY format is the same as for run-coded 3-bit CLUT.

Figure V.31 QHY pixel pair scheme



$i$  = even  
 $H$  = 3 bit QHY code  
 $L$  = 8 bit run length = Number of pixel pairs in run.

where  $2 \leq L \leq 255$  and  $L = 1$  is forbidden.

$L = 0$  means 'Continue this run to the end of the line'.

Every displayed line must finish with a zero-length run ( $L=0$ ). This run must start no later than the last-but-one pixel-pair to be displayed, i.e. it must start no later than pixel-pair 383 for a 625/compatible image, or pixel-pair 359 for a 525 line image. This means that the displayed line must have at least two identical pairs of QHY codes at its right hand end.

#### 4.4.3.2 Decoding Model

The process of decoding the QHY High Resolution mode is shown in Figure V.32. The DYUV Normal Resolution image is decoded as described in V.4.4.2.2 so as to give Normal Resolution  $Y$ ,  $U$  and  $V$  components. These are expanded by the interpolation filter of Figure V.20 to give filtered High Resolution components  $Y_f$ ,  $U_f$  and  $V_f$  which are then matrixed to give filtered High Resolution components  $R_f$ ,  $G_f$  and  $B_f$ .

V.4 The Video Decoder

The runlength encoded QHY data is decoded using the table of quantization values transmitted with the image so as to give the High Resolution difference data.

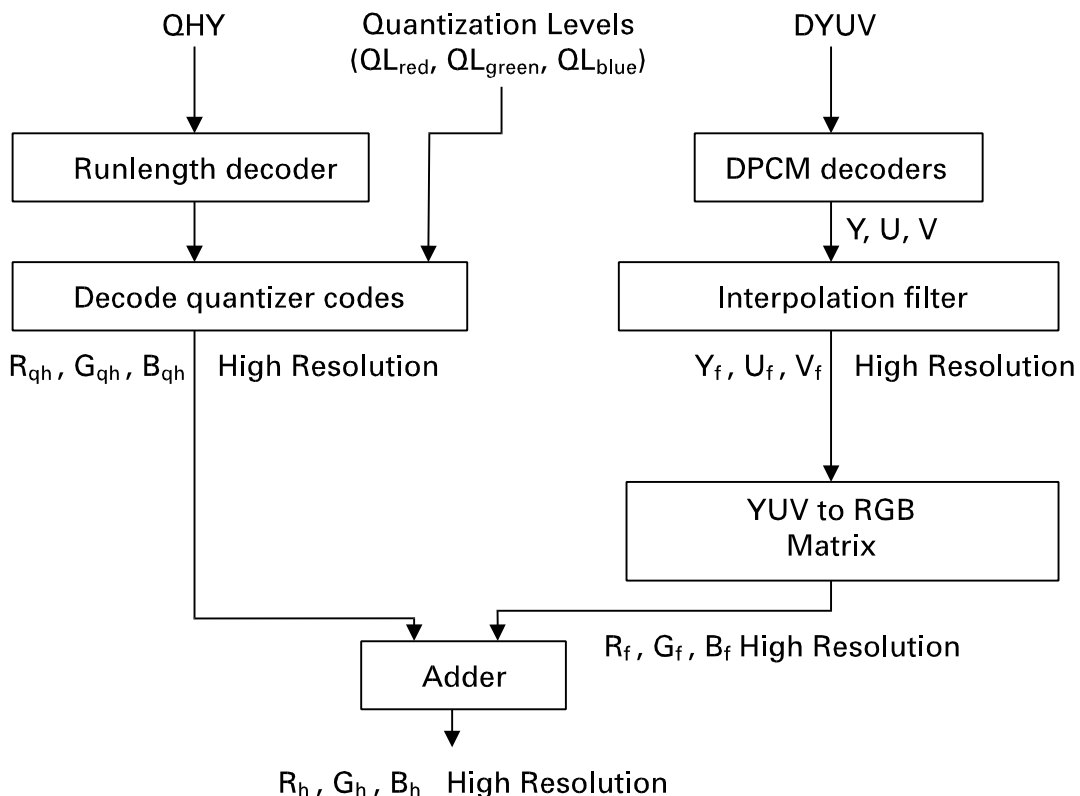
An indirect method of summing the QHY data with the filtered Y data is used. Instead of decoding the QHY codes to a single  $Y_{qh}$  difference value, equivalent RGB difference values  $R_{qh}$ ,  $B_{qh}$  and  $G_{qh}$  are derived which are summed with  $R_f$ ,  $G_f$  and  $B_f$  to give a restored High Resolution image  $R_h, G_h, B_h$ . This operation is equivalent to summing the value  $Y_{qh}$  with the filtered value  $Y_f$  prior to matrixing, provided that

$$R_{qh} = G_{qh} = B_{qh} = Y_{qh}$$

The RGB difference values are given by

$$R_{qh} = (QL_{red} - 128) * 2 \quad G_{qh} = (QL_{green} - 128) * 2 \quad B_{qh} = (QL_{blue} - 128) * 2$$

Figure V.32 QHY High Resolution Mode Decoding Process



### 4.4.3.3 Use of QHY for High Resolution Graphics

Because of the manner in which the QHY data is added to the DYUV data after matrixing to RGB it is possible to optionally use the QHY data for encoding high resolution graphics, simultaneously with its use for high resolution natural pictures. One or more of the QHY codes may be assigned in the decoder to levels which when decoded and added to fixed DYUV decoded levels result in appropriate graphics colours.

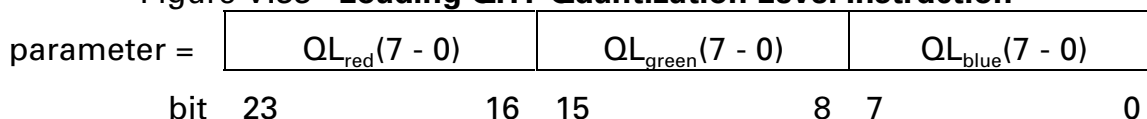
For this application the QHY decoder may be set to give bipolar output values which are **not** equivalent to a matrixed  $Y_{qh}$  value i.e. the  $R_{qh}$ ,  $B_{qh}$  and  $G_{qh}$  values are not equal. Use of this facility enables high resolution natural images to be combined with computer generated graphics and text in a single subscreen.

### 4.4.3.4 Loading QHY Quantization Levels

QHY quantization levels are loaded via the Display Control Program (V.4.5.1). The decoder uses the lowest 8 entries of CLUT bank 2 to hold these levels. The QHY bank (bank 2) of the CLUT must be selected before loading. The instructions are:

#### Load QHY quantization level 0 - 7

Figure V.33 Loading QHY Quantization Level Instruction

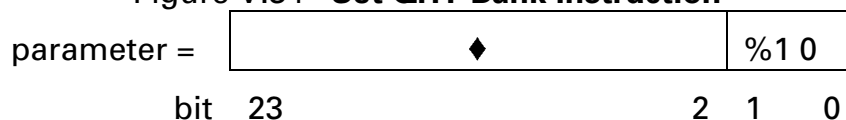


For normal QHY,  $QL_{red} = QL_{green} = QL_{blue} = QL$  (see V.3.5.3), but for special graphics applications the values may differ.

These instructions are identical to the first 8 Load CLUT color instructions for bank 2 (see Figure V.45).

#### Set QHY bank:

Figure V.34 Set QHY Bank Instruction



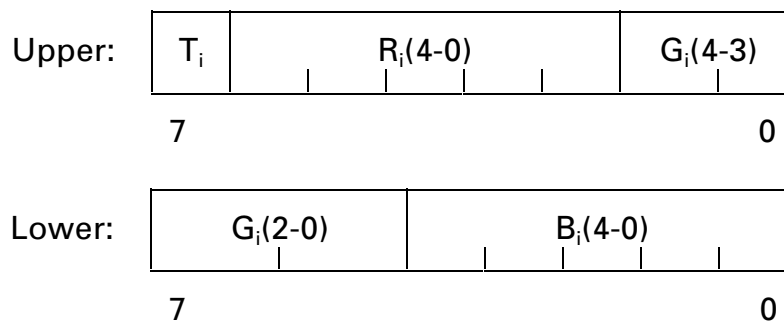
This instruction is identical to the Set CLUT bank instruction (see Figure V.45) for bank 2.

\*\*\*\*\*

**4.4.4 Absolute RGB**

The 16 bit word representing each pixel is split into two halves, upper and lower, the first in image store 0 and the second in image store 1.

Figure V.35 **Memory Representation Absolute RGB**



R, G and B are 5 bit absolute values of Red, Green and Blue respectively. Each component is decoded to the standard form of V.4.4.1.2 by multiplying by 8. A value of 2 represents black level.

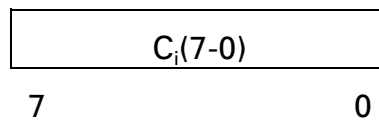
T is the **T**ransparency bit (see V.5.7)

**4.4.5 CLUT**

**4.4.5.1 8-bit CLUT**

The representation in memory is:

Figure V.36 **Memory Representation 8-bit CLUT**

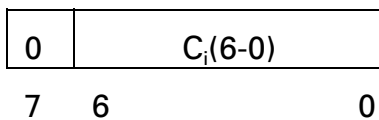


C = 8 bit CLUT address.

**4.4.5.2 7 bit CLUT**

The representation in memory is:

Figure V.37 **Memory Representation 7-bit CLUT**

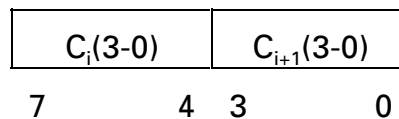


C = 7 bit CLUT address.

**4.4.5.3 4 bit CLUT**

The basic unit is the pixel pair. The representation in memory is:

Figure V.38 **Memory Representation 4-bit CLUT**



where C = 4 bit CLUT address and i must be even.  
Each byte represents a pixel pair starting at an even X coordinate.

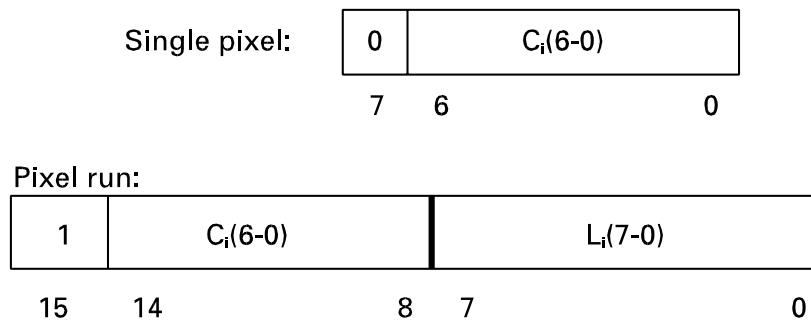
#### 4.4.6 Runlength Coded CLUT

Run-coded images are stored in memory in compressed form, and are expanded to CLUT codes by a runlength decoder associated with each image store (see Figure V.26). There is no linear relationship between pixel number and memory address. Lines of an image are of variable length in memory. The memory representations are given below.

##### 4.4.6.1 Run-length coded 7-bit CLUT

Single pixels are encoded as one byte, and a run of pixels of the same color as two bytes.

Figure V.39 Memory Representation RL 7-bit CLUT



C = 7 bit CLUT address,

L = 8 bit run length = Number of pixels in run

where  $2 \leq L \leq 255$  and  $L = 1$  is forbidden.

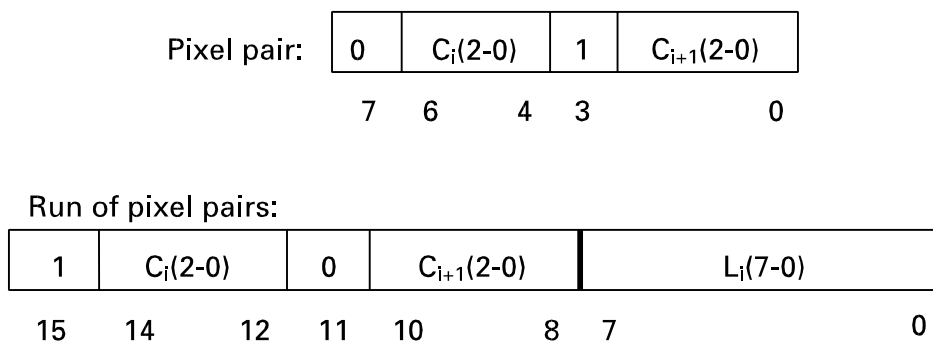
L = 0 means 'Continue this run to the end of the line'.

Every displayed line must finish with a zero-length run (L=0). This run must start no later than the last-pixel-but-one to be displayed, i.e. it must start no later than pixel 383 for a 625/compatible image, or pixel 359 for a 525 line image. This means that the displayed line must have at least two identical pixels at its right hand end.

**4.4.6.2 Run-length coded 3-bit CLUT**

The same basic memory representation is used as for 7-bit CLUT, but with a different interpretation of the pixel code byte. The 3-bit CLUT codes are converted to 4 bit CLUT codes by appending a zero msb before they address the CLUT.

**Figure V.40 Memory Representation RL 3-bit CLUT**



- i = even
- C = 3 bit CLUT address,
- L = 8 bit run length = Number of pixel pairs in run

where  $2 \leq L \leq 255$  and  $L = 1$  is forbidden.

$L = 0$  means 'Continue this run to the end of the line'.

Every displayed line must finish with a zero-length run ( $L=0$ ). This run must start no later than the last-but-one pixel-pair to be displayed, i.e. it must start no later than pixel-pair 383 for a 625/compatible image, or pixel-pair 359 for a 525 line image. This means that the displayed line must have at least two identical pixel-pairs at its right hand end.



V.4 The Video Decoder

---

4.4.7 CLUT Organization

The CLUT has 256 entries. In order to facilitate sharing of the CLUT between the two paths the CLUT is divided into four banks of 64 entries each, Bank 0 to Bank 3. The CLUT usage by the various planes is shown in Figure V.41.

Figure V.41 CLUT Organization

		8-bit CLUT	7-bit CLUT	4-bit CLUT	3-bit CLUT <sup>3</sup>
CLUT Bank 3	63 0	255 • • •	127 • Path 1 Plane B or Path 0 <sup>4</sup> Plane A		
CLUT Bank 2	63 0	• Path 0 Plane A	0	15 Plane B 0	7 Plane B 0
CLUT Bank 1	63 0	• • •	127 • • Path 0 Plane A		
CLUT Bank 0	63 0	• • 0	• 0	15 Plane A 0	7 Plane A 0

---

<sup>3</sup> May be used in path 0 for dual 7-bit CLUT

<sup>4</sup> May be used in path 0 for dual 7-bit CLUT

#### 4.4.8 Allowed Image Coding Combinations

The allowed image coding combinations are determined by:

- (1) The bits/pixel for each plane. These are specified in the basic decoding model.
- (2) The way the CLUT is shared between display paths.
- (3) The data rate available in each path. This determines the allowed resolutions.

The basic set of allowed combinations, from (1) above, is:

Plane A: Off, DYUV, CLUT4, CLUT7, CLUT8, RL3, RL7.

Plane B: Off, DYUV, CLUT4, CLUT7, RL3, RL7, RGB555, QHY.

**Note:**

RL3 is a special case of CLUT 4, with the most significant bit set to 0 by the decoder before CLUT lookup

**Restrictions**

- (1) RGB555 uses both image data streams, so it can only be used with plane A turned off.
- (2) CLUT8 uses the whole CLUT, so it may only be used with plane B set to DYUV or turned off.

The same restriction applies to CLUT7 and RL7 used with dual 7-bit CLUTs.

- (3) QHY is defined only for the extended case, and must be used with Normal Resolution DYUV in plane A.

**4.4.9 Allowed Image Coding Resolutions**

The specified resolutions of each coding type are shown in Figure V.42.

Figure V.42 **Allowed Image Code Resolutions**

	Normal	Double	High
CLUT4	-	B	B
CLUT7	B	-	E
CLUT8	B	-	E
RGB555	B	-	-
RL3	-	B	B
RL7	B	-	E
DYUV	B	-	E
QHY	-	-	E

B = Specified image coding for Base Case

E = Specified image coding for Extended Case

- = Unspecified image coding

It should be noted that application-specific coded data can be transformed by 'software manipulation' to produce an image conforming to one of the above specified coding types. This is allowed within this specification (see V.6.3.1).

Note also that, by use of the interlace (see V.5.14), the vertical resolution of all Normal and Double resolution image formats may be doubled. In the case of Double resolution formats, i.e. CLUT4 and RL3, the result is High resolution.

#### 4.5 Display Control Facilities

Most display control is performed by means of a display control program (DCP) associated with each path (see Figure V.27).

UCM functions (see VII.2.3.4) are provided to access the DCP's.

##### 4.5.1 Display Control Program

A detailed specification of the DCP is given in VII.2.3.2 and a short description is given here.

A display control program consists of a Field Control Table (FCT) and a Line Control Table (LCT). The FCT is a one-dimensional array of instructions, which are carried out before the start of each field. The LCT is a two dimensional array of instructions, each row of which is associated with a display line. Successive rows of the table correspond to successive display lines. The control actions corresponding to the entries in a row are carried out between the end of the previous line and the start of the line with which the row is associated.

The LCT control mechanism allows display parameters and visual effects to be redefined on a line by line basis.

Each instruction consists of a one byte control code followed by three bytes of parameters. The available instructions are shown in the tables given in Figures V.43, V.44 and V.45 and their respective parameters are detailed in the sections listed under 'more details'. **All the codes in these tables may be used in either the FCT or the LCT** unless otherwise stated. All codes not in these tables are reserved.

The Display Control Programs in the two paths operate in conjunction. The functions specific to each path are controlled only by that path. Most global control functions are only available in path 0, but some are available in both paths.

Display parameters loaded by a DCP instruction on a particular line are retained on following display lines, until over-written by a subsequent instruction on a later line. However, display parameters are not guaranteed to be retained from field to field. They are un-defined at the start of a field, and must be loaded afresh for each field.

**Figure V.43 Control Program Instructions available for Path 0 only**

Code	Action	More details
\$C0	Select image coding methods	V 4.6.1
\$C1	Load transparency control information	V 5.7.3
\$C2	Load plane order	V 5.7.1
\$C4	Load transparent color for plane A	V 5.7.2.2
\$C7	Load mask color for plane A	V 5.7.2.2
\$CA	Load DYUV start value for plane A	V 4.6.2
\$D8	Load backdrop color	V 5.13
\$D9	Load mosaic pixel hold factor for A	V 5.11.1.1
\$DB	Load image contribution factor for A	V 5.9

**Figure V.44 Control Program Instructions available for Path 1 only**

Code	Action	More details
\$C6	Load transparent color for plane B	V 5.7.2.2
\$C9	Load mask color for plane B	V 5.7.2.2
\$CB	Load DYUV start value for plane B	V 4.6.2
\$DA	Load mosaic pixel hold factor for B	V 5.11.1.1
\$DC	Load image contribution factor for B	V 5.9

Figure V.45 **Control Program Instructions available for Both paths**

Code	Action	More details
\$10	No operation	-
\$20	Load control table line start pointer	Note 1
\$40	Load display line start pointer	V 4.5.2.2
\$60	Signal when scan reaches this line	V 5.6
\$78	Load display parameters	V 4.6.1
\$80-\$BF	Load CLUT color 0-63 (of current bank)	V 5.5
\$C3	Set CLUT bank	V 5.5
\$D0-\$D7	Load matte register 0-7	V 5.10.3
\$80-\$87	Load QHY level 0-7	V 4.4.3.4
\$C3	Set QHY bank	V 4.4.3.4

**Note 1**

This instruction may only be written by means of the UCM calls DC\_LLnk and DC\_FLnk. It is the responsibility of the application to insure that space is left in the LCT or FCT by means of this mechanism before executing the DC\_LLnk or DC\_FLnk commands.

The video driver will place this instruction in the last column of the LCT when using the DC\_LLnk command. In the case of DC\_FLnk, the video driver will place this instruction beyond the end of the application FCT.

---

<sup>5</sup> This action may be set independently in each path.

## 4.5.2 Line Pointer Tables

### 4.5.2.1 Image Line Pointer Table

Each line of an image as stored in image memory may start at an arbitrary address within the available image memory. It is not necessary for lines to be contiguous in memory. A table of line start addresses, or line pointer table, may be maintained for each image (associated with each drawmap, see VII.2.3).

This table may be used for example for line repeat (for e.g. error concealment), or for partial updates of run-coded images.

### 4.5.2.2 Display Line Start Pointers

The mechanism for displaying an image is the Display Line Start Pointer instruction in the DCP. These instructions may go into the FCT or any line in the LCT, however there must be one (and only one) DLSP instruction in the combined area of the FCT and the first line of the LCT linked to by the FCT. There is no restriction on the number or placement of DLSP instructions outside of this combined area.

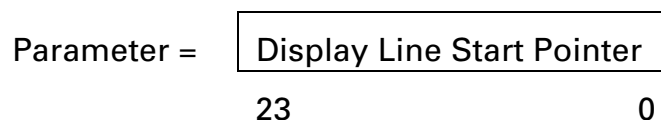
If there is no display line start pointer for a line, the line will start at a memory address directly after the end of the previous line.

The display line start pointers may be derived from the image line pointer table(s) of the image(s) being displayed on that path. Offsets are added for image positioning, and effects such as vertical mosaics and subscreens may be produced.

The display line start pointer is inserted into the DCP by the instruction given below.

#### Load Display Line Start Pointer:

Figure V.46 Load Display Line Start Pointer Instruction



## 4.6 Display Control Functions

A few basic display control functions are detailed in this section. Those concerned with visual effects are treated in V.5.

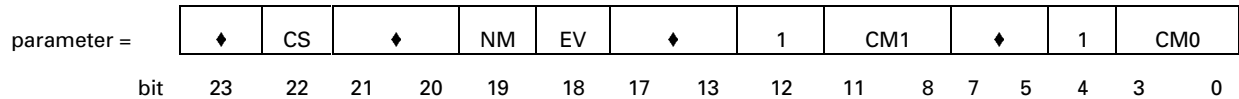
### 4.6.1 Selecting Image Coding Methods

The image coding methods decoded by the video decoder are chosen by the two following DCP instructions.



**Select Image Coding Methods:**

Figure V.47 **Select Image Coding Methods Instruction**



where CS = CLUT select for dual 7-bit CLUTs

- 0 = Banks 0 and 1
- 1 = Banks 2 and 3

CM0 = Coding Method for path 0

CM1 = Coding Method for path 1

Figure V.48 **Coding Values CM0 and CM1**

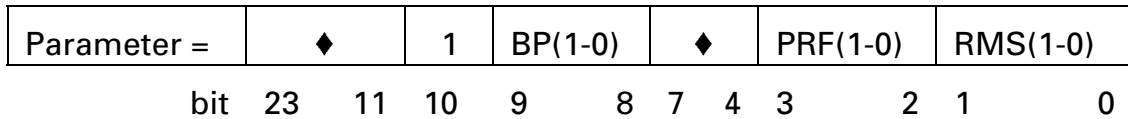
Values	CM0	CM1
0000	OFF	OFF
0001	CLUT8	RGB555
0011	CLUT7 or RL7 (Single CLUT)	CLUT7 or RL7
0100	CLUT7 or RL7 (Dual CLUT)	-
0101	DYUV	DYUV
1011	CLUT4 or RL3	CLUT4 or RL3
1111	-	QHY (Extended case)

Values are in binary notation  
All other values are reserved.

- NM = Number of Mattes
  - 0 = 1 matte
  - 1 = 2 mattes
  
- EV = External Video Enable
  - 0 = Disable
  - 1 = Enable

**Load Display Parameters:**

Figure V.49 **Load Display Parameters Instruction**



where RMS = Runlength or Mosaic (Pixel Repeat) Select

- where 00 = Normal
- 01 = Normal
- 10 = Runlength
- 11 = Mosaic

PRF = Pixel Repeat Factor for Mosaics

- where 00 = times 2
- 01 = times 4
- 10 = times 8
- 11 = times 16

BP = Bits/Pixel from the Image Store

- where 00 = 8 bits/pixel, Normal Resolution.
- 01 = 4 bits/pixel, Double or High (\*) Resolution.
- 10 = 8 bits/pixel, High (\*) Resolution
- 11 = Reserved
- \* = Extended Case only.

**Note:** Parameter values are given in binary notation.

Only those combinations of CM0, CM1, BP and RMS that are defined image combinations according to the CD-I specification are allowed. All others are reserved. The allowed combinations are shown in Figure V.50 and V.4.4.8.

Figure V.50 **Allowed Image Coding Method Combinations**

	CM0 or CM1, BP, RMS Binary Values)		
	Normal	Double	High
CLUT4	-	1011,01,00	1011,01,00
CLUT7	0011,00,00	-	0011,10,00
CLUT8	0001,00,00	-	0001,10,00
RGB555	0001,00,00	-	-
RL3	-	1011,01,10	1011,01,10
RL7	0011,00,10	-	0011,10,10
DYUV	0101,00,00	-	0101,10,00
QHY	-	-	1111,01,00

6

6

Each entry in this table is in the form CM0 (or CM1), BP, RMS. Note that, for CLUT and RGB coding methods, RMS may also be set to %11, to select mosaic pixel repeat.

It should be noted that applications specific coded data can be transformed by 'software manipulation' to produce an image of one of the above specified coding types. This is allowed within this specification (see V.6.3.1).

---

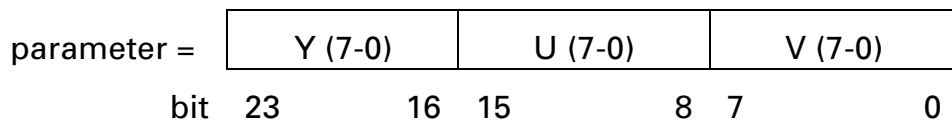
<sup>6</sup> For dual 7-bit CLUT, 0011 must be replaced by 0100

**4.6.2 Loading DYUV Start Values**

The Absolute YUV starting values for a line of the displayed part of an image are loaded by the two DCP instructions one for plane A and one for plane B.

**Load DYUV Start Value (for plane A or plane B):**

Figure V.51 **Load DYUV Start Value Instruction**



where Y = Absolute Y start value  
 U = Absolute U start value  
 V = Absolute V start value

If, on a particular line, this instruction is absent, the value given by the last Load DYUV Start Value instruction is used by the decoder.

## 4.7 Error Concealment

### 4.7.1 General

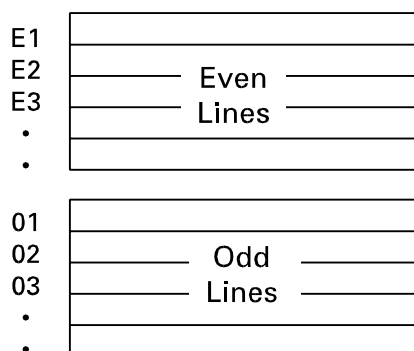
As video pixel data is coded on disc in Form 2 sectors, there is no extra layer of error correction for video beyond the CIRC (see II.4.6). The option is therefore available of using Error Concealment, (a general term for any method of masking the, in this case, visual effect of errors in an image). Error concealment is made possible by the high degree of spatial correlation in most images, especially natural images.

The Form 2 error detection mechanism is capable of signalling that errors have been detected in specified 8-bit bytes or 16-bit words (depending on the implementation) of a video data stream. The player can thus detect which lines of an image are in error, and carry out an appropriate concealment strategy.

### 4.7.2 Odd/Even Line Separation

When error concealment is to be used, the even and odd lines of an image are coded on disc in separate sectors, and are physically separated by interleaving so as to minimize the chance of an error occurring in adjacent lines (see V.6.4.3). In the decoder, the odd and even lines are loaded into two areas in memory, as shown in the figure below.

Figure V.52 **Odd/Even Line Separation in Memory**



The Create Drawmap Function (DM\_Creat) of the UCM (see VII.2.3.4.1) must be called with the line interleave parameter set to 1 in order to set up the line pointer table for the image correctly. If error concealment is not to be used, the interleave parameter is set to 0, and the lines of the image are consecutive on disc and in memory.

### 4.7.3 Concealment Techniques

The concealment strategy chosen will depend on the time available. The longer an image is to be displayed, the more noticeable errors are, and the more time is available for concealment. Some simple techniques are defined here.

(1) Natural Images (DYUV)

When an error is detected, the line containing the error is substituted by the previous line. The contents may be copied across, or the line start pointer in the DCP for the line in error may be substituted by that of the previous line. If the first line is in error the next line is taken.

(2) CLUT or RGB555 Images

When an error is detected, the pixel in error is replaced by the adjacent pixel in the previous line. If the pixel is in the first line then the corresponding pixel of the next line is taken.

(3) Runlength Images

When an error is detected, the line containing the error is substituted by the previous line. The lines will probably not have the same length in memory, so direct copying is undesirable. The line start pointer in DCP may be used to achieve the same effect, as defined above for natural images. If the first line is in error then the next line is taken.

### Error concealment of DYUV images

Any line of a DYUV image that is replaced according to the above method must also have its absolute YUV start value copied from that of the line that is replacing it.

#### 4.8 525/625 Line Image Interchange

All CD-I images may be displayed on any decoder. To enable this, the video decoder is capable of changing the dimensions of the full-screen display area (while leaving the pixel clock frequency the same) according to the state of a flag called 'Compatibility Mode'. This can take the values 0 and 1, and is set by the UCM function DC\_SetCmp. The value 0 corresponds to the display dimensions specified in V.2.2.

The dimensions of the full-screen display in the two cases, and the offsets in pixels of the top left corner of the display for Mode = 1 from that for Mode = 0, are given in Figure V.53 below. Dimensions are given as width followed by height. A positive offset indicates displacement downwards or to the right.

Figure V.53 525/625 Line Image Interchange

Compatibility Mode	525 Monitor	525 line TV	625 line system
0	360 x 240	384 x 240	384 x 280
1	384 x 240	360 x 240	360 x 240
Vert. offset	0	0	+20
Horiz. offset	-12	+12	+12

The vertical offsets are mandatory. The horizontal offsets are strongly **recommended**, to ensure visibility of the safety area. The actual offset is returned by DC\_SetCmp: it is needed by the application in order to be able to set the pointer offset.

In the case of 525 TVs and 625 line systems, the Mode 1 display does not necessarily extend beyond the screen boundary - however, the term 'full-screen display' is retained for consistent usage.

In the case of 525 monitors, the Mode 1 full-screen display width is such that the extreme edges may not actually be displayed. However, the full width is read from memory for each line.

See Appendix V.2 for guidelines on how the application may use this facility to achieve the goal of correctly displaying all images on all decoders.

## V.5 Visual Effects

### 5.1 General

A number of visual effects are possible with the CD-I system, which control or affect the way that images are displayed and combined. These effects are defined in this section.

The basic visual effects facilities available via the Display Control Program and accessed via the UCM (see VII.2.3.2.4) are detailed. High level functions based on these are also described.

The effects specified in this section are not an exhaustive set. An unlimited number of further application-specific effects, based on software manipulation of the image, or on use of the basic effects facilities described here, are also possible.

Timing and synchronization of video information is not covered in this chapter, except for synchronization to display scanning. Refer to Chapter VII (CD-RTOS) for details on these topics.

### 5.2 Cut

The most basic effect in film and video editing is called 'Cut'. 'Cut' causes an image to appear, usually to replace a previous image.

'Cut' may be achieved either by enabling an image for display on a given plane, or by re-ordering the planes to bring an image to the front.

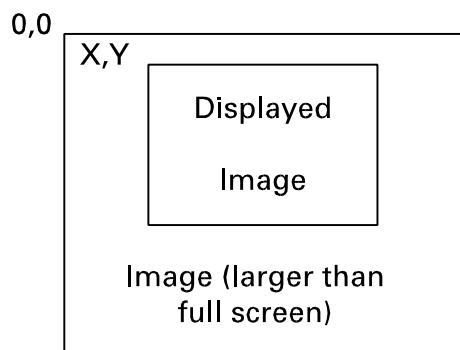


### 5.3 Scroll

#### 5.3.1 Image Positioning

Images larger than the full-screen dimensions may be encoded and loaded by the player. It is possible to position the displayed part of such images at any coordinate X,Y relative to the image top left corner.

Figure V.54 **Positioning of Displayed Image within Larger Image**



For DYUV coded images, it is necessary to reload the absolute YUV starting values for each line whenever an image is re-positioned. These may be calculated from the old values and the image contents. For DYUV images, X must be a multiple of two pixels.

For runlength coded images the horizontal position must be at the left-hand edge, i.e.  $X = 0$ . The same applies to high-resolution DYUV+QHY images, as the QHY component is run-coded. (However, applications may arbitrarily position run-coded and QHY images by manipulation of the codes).

**5.3.2 Scroll**

Scroll is the repeated re-positioning of a displayed image within a larger image. Motion may be vertical, horizontal, or a combination of the two. A wide range of scrolling effects may be achieved by use of image re-positioning.

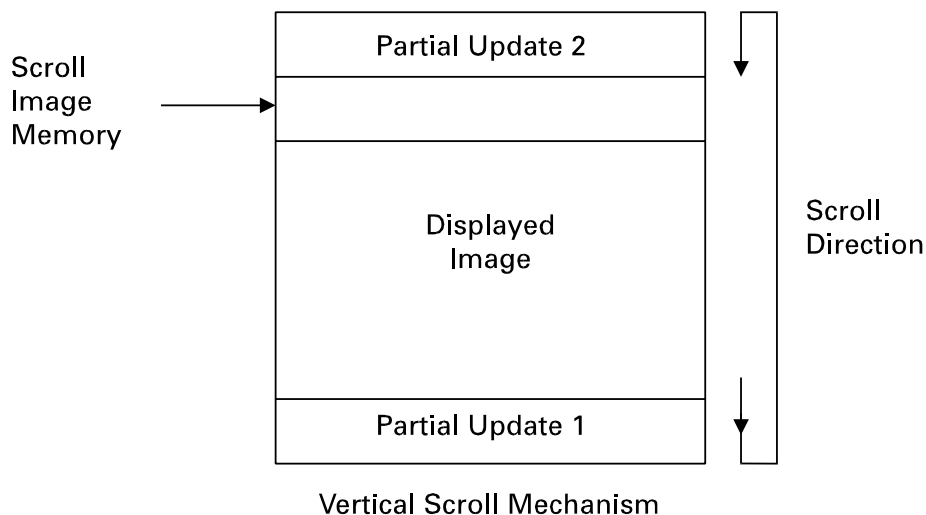
For a scroll which is constrained to be within the boundaries of an image, larger than screen size, but which is small enough to be held in decoder memory at one time, the simple re-positioning mechanism is all that is required.

However, if the illusion of scroll over a very large image is required, such as panoramic scan of a landscape, or vertical scroll of a multipage document, the technique used in the figure below must be used.

A Scroll Image is defined in memory, larger than full-screen. The displayed image is repeatedly re-positioned within the larger image, and when it reaches the edge it is broken into two parts at opposite ends of the Scroll Image (see Figure V.55). The line pointer table is manipulated to form the two parts into one displayed image on the screen. Partial updates are progressively performed to write new information to the image in the direction of scroll.

A similar technique can be used in the horizontal direction.

**Figure V.55 Vertical Scroll Mechanism**



### 5.3.2.1 Scroll of DYUV Images

When a DYUV image is moved horizontally, new absolute start values must be loaded into the DCP for each line. These may be calculated by the application from the old values and the image contents.

This method will suffice for slow scroll speeds, but for high speeds the start values must be loaded from disc. Horizontal scroll must be by multiples of two pixels.

### 5.3.2.2 Scroll of Runlength Images

Because of the properties of runlength coded images, horizontal scrolling is possible only by application-specific manipulation of the run-codes. Vertical scroll is possible, in the same way as for other coding types.

The same comments apply to DYUV+QHY encoded High Resolution images.

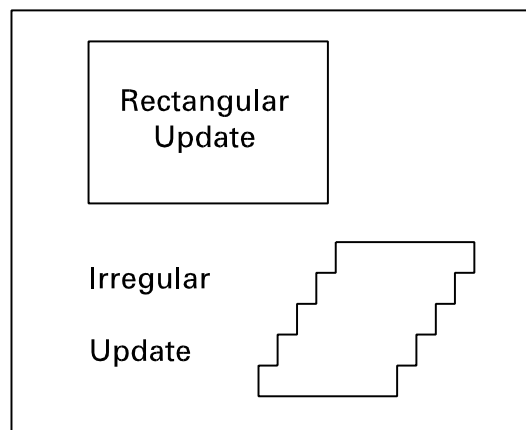
Scrolling effects may also be achieved for runlength animation by loading from disc successive images representing a scene with increasing displacement.

### 5.4 Partial Update

It is possible to encode and load images which are smaller than full-screen. These small images cannot be displayed directly, but only by copying them over a part of an image large enough to display. Such alterations to an image are known as Partial Updates.

One of the main uses for Partial Updates is to create visual motion in a limited area of the screen by repeatedly updating that area. Visually attractive piece-by-piece transitions from one image to another may also be achieved.

Figure V.56 **Example Rectangular and Irregular Partial Updates**



Two types of partial updates are available, for rectangular areas and irregular areas.

#### 5.4.1 Rectangular Updates

A rectangular area is coded as a normal image, of reduced size. It is then copied in to the displayed image at the desired coordinate (see UCM, VII.2.3).

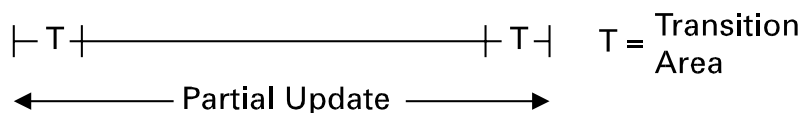
#### 5.4.2 Irregular Updates

The shape of an Irregular Update is coded as a Y starting value then an X position and length for each successive line of the update. The pixel data itself is packed in line order with no gaps between lines. The UCM function that performs an irregular update is DM\_IrWr (see VII.2.3.4.1).

### 5.4.3 Partial Update of DYUV Images

As DYUV pictures are differentially coded, it is necessary to begin each line of a Partial Update with a 'transition area'. This is a short sequence of DYUV codes which have been pre-calculated to make the transition from the YUV values in the surrounding image just preceding the update to those at the left edge of the update proper.

Figure V.57 **Partial Updates Transition Areas**



Each line of a Partial Update must similarly be terminated with a transition area. Both transition areas must form part of the Partial Update as coded on disc.

If the Partial Update includes the beginning of any lines in the display screen, then the absolute starting value for those lines must also be loaded from disc.

For High Resolution DYUV+QHY images, the QHY component is run-length coded, and therefore Partial Updates of this component must use the techniques described below for runlength images.

### 5.4.4 Partial Update of Runlength Images

Because of the properties of runlength coding, Partial Updates of runlength coded images are only specified for Rectangular Updates with the full width of the image. However, arbitrary Partial Updates may be carried out by application-specific manipulation of the run-codes or by prearranging the run lengths on disc, to accommodate the Partial Updates.

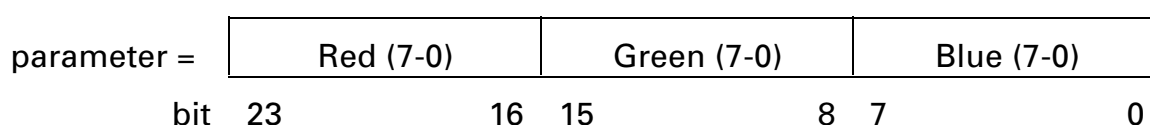
A runlength update generally will not have the same length (in bytes of code) as the part of the image it is replacing. To avoid constantly moving the part of the image below the update, the line pointer table may be used to provide room for the largest update, and to join up the parts of the image.

### 5.5 CLUT Update

CLUT values are loaded via the Display Control Program (V.4.5.1). Colors are loaded on a bank basis, i.e. the correct bank must be selected before loading. The instructions are given below.

#### Load CLUT color (0 - 63 of current bank):

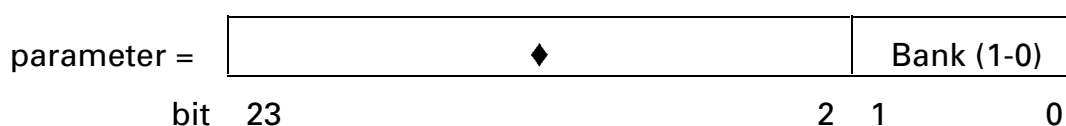
Figure V.58 Load CLUT Color Instruction



and

#### Set CLUT bank :

Figure V.59 Set CLUT Bank Instruction



The Red, Green and Blue components are directly in the form to which all images are decoded, i.e. they have black level at 16 and nominal peak level at 235 (see V.4.4.1.2).

The ability to load the CLUT banks depends upon the active Image Coding Method (V.4.6.1) in the DCPs. In general, CLUT banks 0 and 1 must always be loaded from path 0. When path 0 is set to CLUT8, banks 2 and 3 must be loaded from path 0. Also, when path 0 is set to CLUT7 or RL7 and the CLUT Select bit is set to path 1 (dual 7-bit CLUT), banks 2 and 3 must be loaded from path 0.

When path 1 is set to CLUT4, CLUT7, RL3 or RL7 banks 2 and 3 may be loaded from path 1. If path 1 is not set to one of these modes, then banks 2 and 3 may be loaded from either path 0 or 1.

#### 5.5.1 CLUT Animation

For CLUT coded images, the 256 values in the CLUT control the colors of the entire image. Therefore, some simple animation effects are possible simply by redefining some or all of the CLUT contents as a function of time.

### 5.5.2 Dynamic CLUT Update

By reloading CLUT colors on successive lines of the display, it is possible to use more colors in an image than the number of entries in the CLUT.

The required CLUT loading commands are placed in the appropriate lines of the Line Control Table.

It is of course possible to use this effect in conjunction with CLUT animation.

### 5.5.3 Dual 7-bit CLUTs

The 7-bit CLUT coding methods, that is CLUT 7 and RL 7, may be used with dual 7-bit CLUTs. This function, available in path 0 only, allows either the lower or upper half of the CLUT to be used for display, selectable on a line by line basis.

Dual 7-bit CLUT operation is selected via the CM0 field of the DCP instruction 'Select Image Coding Methods' (see V.4.6.1). Which half of the CLUT to use for display is selected by the CS field of the same instruction.

Dual 7-bit CLUTs may be used e.g. to increase the number of colors in a runlength coded image, or to provide two different sets of colors for split-screen operation.

As the dual 7-bit CLUT function allocates both CLUTs to path 0, it cannot be used simultaneously with CLUT or RL coding in path 1.

### 5.6 Synchronization to Display Scanning

If an object is repeatedly being drawn on the screen, erased, and then drawn at a new position, such as in some computer graphic applications, a flicker effect is frequently seen. This is caused by the object not being fully re-written at the time it is scanned for display.

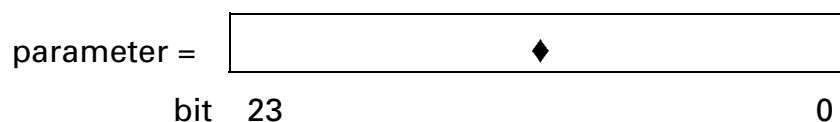
This effect should be avoided by synchronizing the re-writing of objects to the field scan, so that erasure and re-writing takes place between successive scans of the object.

It is also necessary to synchronize Partial Updates to field scan, for similar reasons.

A function is provided to generate a video interrupt when a particular scan line (chosen by the application) has been reached by the display scanning mechanism. The Display Control Program instruction used by this function is:

#### Signal when scan reaches this line

Figure V.60 Interrupt at Scan Line Instruction







V.5 Visual Effects

---

**5.7.2 Transparency Mechanisms**

Three mechanisms are defined to control transparency between planes. These are given below.

**5.7.2.1 Transparency Bit**

In the case of absolute RGB coding (RGB555) the most significant bit of every pixel is used for transparency control.

**5.7.2.2 Color Key**

A CLUT (not RGB555 or DYUV) image may be compared on an RGB basis, pixel by pixel, to a color key value K. The result of the comparison (Color Key = TRUE or FALSE), may be used to affect the image transparency.

In order to allow matching for a range of colors, the result of the comparison may be masked on a bit by bit basis with a mask color M. A pixel of color P has color key value TRUE if and only if

$$[P(i) = K(i)] \text{ or } [M(i) = 1]$$

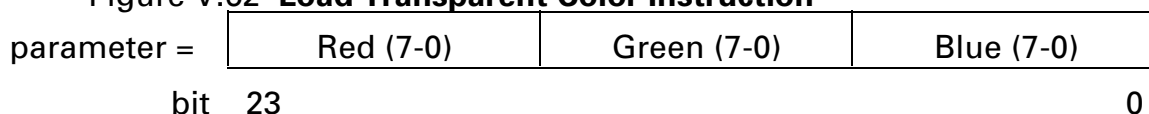
for all bit positions  $i = 2 - 7$  of each color component (R,G,B).

Note that only the most significant 6 bits of each color component are used.

The color key and mask colors may be set separately for the two image planes. They are set by the Display Control Program instructions. Both the transparent and mask color must be set when using the color key mechanism.

**Load Transparent Color (of Plane A or B):**

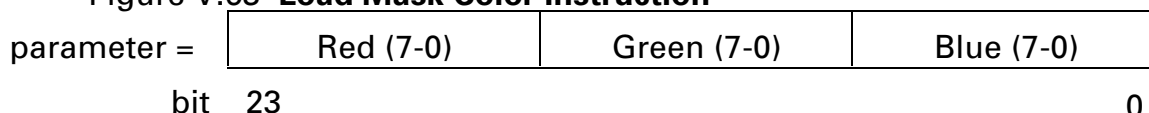
Figure V.62 Load Transparent Color Instruction



and

**Load Mask Color (of Plane A or B):**

Figure V.63 Load Mask Color Instruction



### 5.7.2.3 Transparency via Mattes

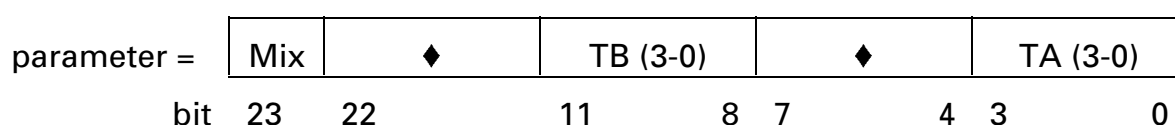
The transparency of an image plane may be linked to the state of a matte flag (see V.5.10).

### 5.7.3 Transparency Control

The three transparency mechanisms described above may be selected individually or in combination, for each of the two image planes. The selection is made via the Display Control Program instruction.

#### Load Transparency Control Information:

Figure V.64 Load Transparency Control Instruction



where Mix = Disable mixing (V.5.9.1).

0 = mix

1 = no mix

TA, TB = Select transparency mechanisms for planes A and B individually.

#### TA, TB<sup>7</sup> Pixel transparent if

0000 Always (plane disabled)

0001 Color key = TRUE

0010 Transparent bit = 1

0011 Matte Flag 0 = TRUE

0100 Matte Flag 1 = TRUE

0101 Matte Flag 0 = TRUE or Color Key = TRUE

0110 Matte Flag 1 = TRUE or Color Key = TRUE

0111 Reserved

1000 Never (no transparent area)

1001 Color Key = FALSE

1010 Transparent bit = 0

1011 Matte Flag 0 = FALSE

1100 Matte Flag 1 = FALSE

1101 Matte Flag 0 = FALSE or Color Key = FALSE

1110 Matte Flag 1 = FALSE or Color Key = FALSE

1111 Reserved

---

<sup>7</sup> Binary values

## 5.8 Wipes

A wipe is the replacement of one image by another during a period of time, by the motion of a boundary separating the visible parts of the two images. The images may both be stationary during this process or may move, for example, with the boundary thus, scrolling in or out.

### 5.8.1 Two-plane Wipes

The chief mechanism for achieving two-plane wipes is that of matte definition. By linking the transparency of the overlaying plane to a matte flag, and re-defining the matte boundary as a function of time, wiping effects of arbitrary complexity can be obtained.

The mechanism for matte definition is described in V.5.10.

A scrolling wipe may be obtained e.g. by scrolling an image at the same time as updating the matte definition.

### 5.8.2 Single-plane Wipes

A wiping effect can be obtained in a single image plane by means of Partial Updates. Wipes of DYUV coded images done this way will not have such clearly defined edges as two-plane wipes, due to the transition area between images.

**5.9 Image Contribution Factors**

Each path has associated with it an Image Contribution Factor, which determines the contribution of its associated image to the final display. This factor is applied after decoding to RGB.

For each pixel, the resultant color components are given by:

$$C = ICF * (C'-16) + 16$$

where ICF = Image Contribution Factor (between 0 and 1)

C = One of the color components, R G or B.

C' = The corresponding component after decoding.

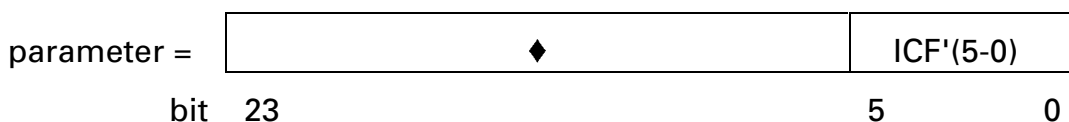
Note that this equation preserves the black level of 16.

ICF's may be used with overlay, or with mixing (see V.5.9.1).

The image contribution factors may be loaded by use of the two Display Control Program instructions, one for plane A and one for plane B.

**Load Image Contribution Factor (for plane A or B):**

Figure V.65 Load Image Contribution Factor Instruction



where

ICF' = 63 \* ICF to the nearest integer between 0 and 63.

i.e. ICF' = 63 gives an image contribution factor of 1.

These instructions allow the ICF's to be changed line by line and continue for subsequent lines unless changed by a new Load Image Contribution Factor instruction. The ICFs may also be changed at arbitrary points on a line by use of the matte mechanism (V.5.10).

### 5.9.1 Image Mixing

As an alternative to overlay, the images on planes A and B may be mixed, that is their components summed.

When mixing is enabled, the color components of each pixel are given by:

$$C = C(A) + C(B) - 16$$

where

$C$  = one of the color components, R, G or B  
 $C(A)$ ,  $C(B)$  = the corresponding components of planes A and B, after application of image contribution factors.

The black level offset of 16 is preserved by this summation.

The result of the mixing function, for each color component, is prevented from overflow or underflow by clipping to 255 and 0 respectively.

To enable/disable mixing, The Display Control Table command 'Load Transparency Control Information' is used (see section V.5.7.3).

Transparency, (see V.5.7.2), may be used with mixing, but in this case transparent areas of an image simply give no contribution to the final display - that is they are equivalent to black areas.

### 5.9.2 Fades and Dissolves

A fade is a gradual decrease or increase of the brightness of an image. A dissolve is the simultaneous fade down of one image and the fade up of a second image mixed with the first.

Dissolves and fades are the result of repeated definition of the Image Contribution Factors as a function of time. The transition characteristics (transition rate, transition law, etc.) are under program control.

Global dissolves and fades are produced using the Load ICF commands (see V.5.9.1). Local dissolves and fades are produced by means of the matte commands, which may be used to change the ICF's at arbitrary points along a line (see V.5.10.2).

### 5.10 Mattes

A matte is any connected area of the display. It may be simply connected (i.e. without holes) or multiply connected (with holes). A matte is used to control the transparency of one or more image planes (V.5.7.3). There may be multiple mattes on the screen at any one time, and some of these may overlap.

The matte mechanism is also used for dynamic control of Image Contribution Factors.

#### 5.10.1 Matte Mechanism

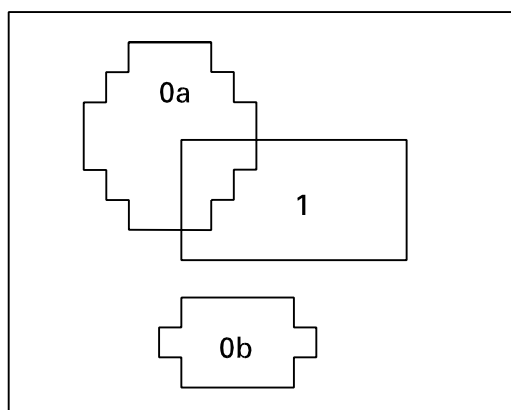
The operation of the matte mechanism in a decoder may be explained in terms of the following model.

There are two matte flags in a decoder, MF0 and MF1, which may be set to TRUE and reset to FALSE at arbitrary positions on the display. A matte flag is TRUE to indicate the inside of a matte, and FALSE to indicate the outside of a matte. Each matte flag may be used to indicate multiple non-overlapping mattes. On the other hand, mattes indicated by different flags may overlap.

Figure V.66 shows an example of overlapping and non-overlapping mattes.

In this figure, mattes 0a and 0b are indicated by flag MF0 and matte 1 is indicated by flag MF1.

Figure V.66 **Example of Overlapping and Non-overlapping Mattes**





### V.5 Visual Effects

---

Setting and resetting of the matte flags is performed via  $N$  matte control registers, MCR0 to MCR(N-1) (in the Base Case,  $N=8$ ). Each of these registers may hold a matte control command, which consists of a position (distance from the left hand side of the screen) at which the command is to be implemented, an operation code, and some parameters.

The matte flags are automatically reset to FALSE at the start of each line.

The commands contained in the registers are carried out on every horizontal image scan line. As each scan line progresses, at each pixel position the lowest-numbered register whose command has not yet been implemented is examined for a position match, and if a match is found that command is implemented.

By means of the NM (Number of Mattes) bit in the 'Select Image Coding Methods' command (V.4.6.1), the  $N$  registers may be divided into two sets of  $N/2$  registers. The two halves operate simultaneously, each in the manner described above, allowing the two matte flags to be changed at the same position.

V.5 Visual Effects

---

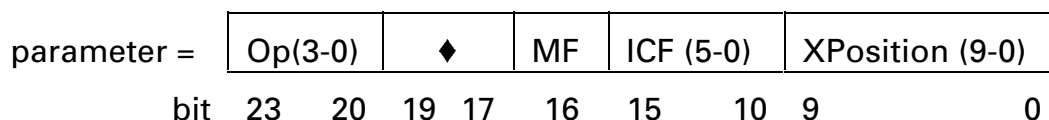
5.10.2 Matte Commands

The matte commands contained in the matte control registers allow setting and resetting of the matte flags MF0 and MF1, and also (independently) the dynamic setting of Image Contribution Factors (ICF's) for the two paths.

An ICF may be changed either at the same position as a matte flag is changed, or independently. The format of Matte Commands is given below.

**Matte Command**

Figure V.67 **Format Matte Command**



where

- XPosition = Distance from left hand edge of display, in Double Resolution
- ICF = Image Contribution Factor
- MF = Matte Flag to be changed<sup>8</sup>
  - 0 = MF0
  - 1 = MF1
- Op = Operation code

**Operation Code**  
(Binary values)

**Action**

0000	Disregard all commands in higher registers
0100	Change ICF for path 0
0110	Change ICF for path 1
1000	Reset a matte flag to FALSE
1001	Set a matte flag to TRUE
1100	Reset a matte flag to FALSE and change ICF of path 0
1101	Set a matte flag to TRUE and change ICF of path 0
1110	Reset a matte flag to FALSE and change ICF of path 1
1111	Set a matte flag to TRUE and change ICF of path 1

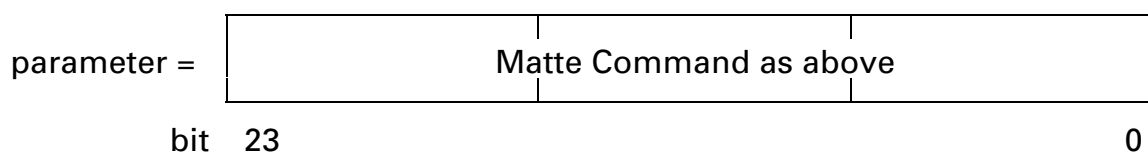
---

<sup>8</sup> The MF bit is only used when the NM (Number of Mattes) bit in the 'Select Image Coding Methods' command (V.4.6.1) is set to 0 (one matte). The values of MF must be **the same** (i.e. 0 or 1) for every matte register.

When NM = 1 (two mattes), matte registers MCRO to MCR3 correspond to matte flag 0 and MCR4 to MCR7 correspond to matte flag 1. The MF bit is ignored.

**5.10.3 Loading the Matte Control Registers**

The contents of the Matte Control Registers remain the same from line to line unless loaded via the Display Control Program. The DCP instructions to load new commands into these registers (one for each register) are

**Load Matte Control Register (0-7):****Figure V.68** Load Matte Control Register Instruction

**Note:** If one or more Matte Control Registers are accessed simultaneously from path 0 and path 1, the access via path 0 has priority over path 1 (access via path 1 will be ignored).

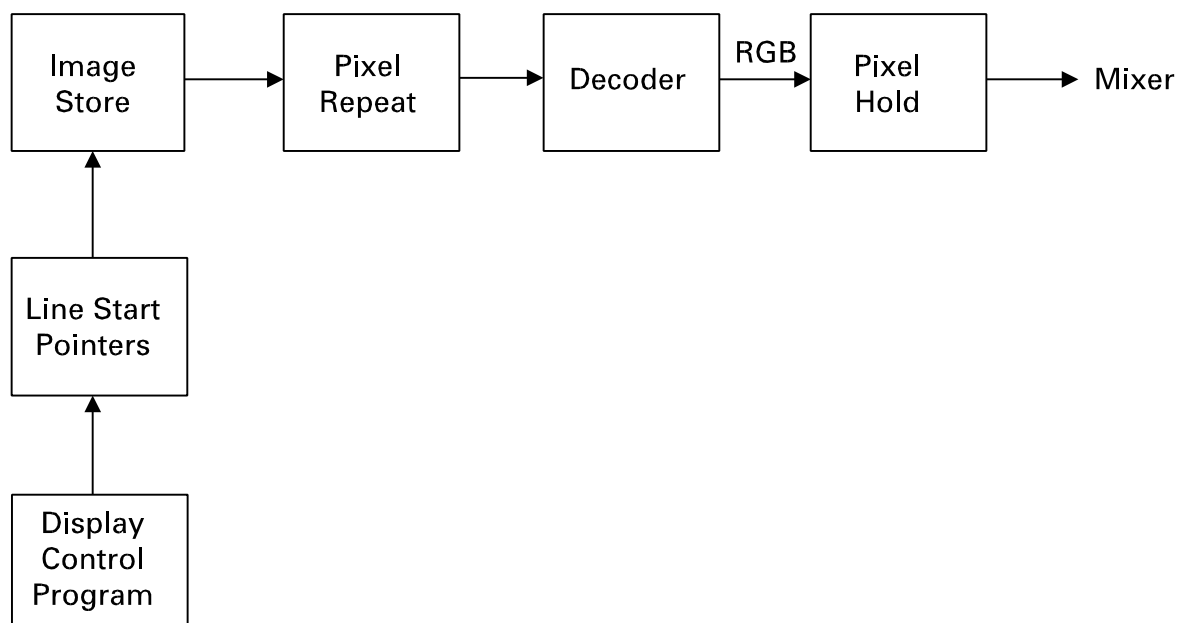
## 5.11 Mosaics

The mosaic functions allow images to be viewed at reduced resolution, either purely as a visual effect, or to allow animation of larger screen areas, or to magnify an image in order to see its detailed structure.

### 5.11.1 Basic Facilities

The three basic facilities that may be used to create mosaic effects are Pixel Hold, Pixel Repeat, Line Hold/Repeat via the Display Control Program. These facilities are shown in Figures V.25 and V.26 but for clarity they are shown again in the Figure V.69 below.

Figure V.69 **Generating Mosaic Effects**



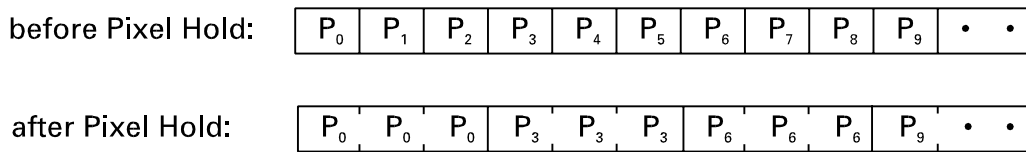
V.5 Visual Effects

---

5.11.1.1 Pixel Hold

The Pixel Hold function operates after the pixel codes have been decoded to RGB. Every Nth pixel RGB value is held for that pixel and the next N-1 pixels, where N is the Pixel Hold Factor. As an example, for N = 3, and where  $P_i$  = the RGB value of pixel i,

Figure V.70 Example of Pixel Hold Function

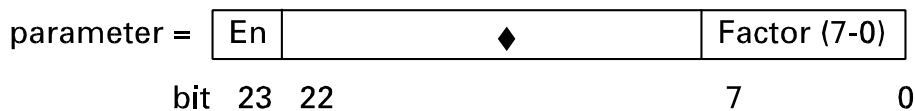


The Pixel Hold function causes a 'Granulation' effect in the horizontal direction, i.e. the resolution is lowered without the image size changing. As Pixel Hold operates after decoding, it can be used for all images however they are coded, either by DYUV, CLUT, RGB or RL.

The Pixel Hold Factor can be any value 2 to 255 and is loaded by means of the DCP instruction given below.

**Load Mosaic Pixel Hold Factor (for plane A or B):**

Figure V.71 Load Mosaic Pixel Hold Factor Instruction



where

- En = Pixel Hold Enable
- 0 = Mosaic off
- 1 = Mosaic on

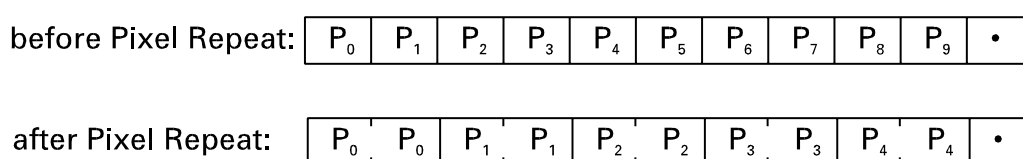
- Factor = Number of pixel periods that the pixels are held
- %00000000 = Reserved
- %00000001 = Normal usage
- %00000010 to %11111111 = Mosaic effect

The pixel hold factor is effective at both Normal Resolution and Double or High Resolution.

### 5.11.1.2 Pixel Repeat

The Pixel Repeat function repeats codes from the image memory, as distinct from holding onto decoded color values. Each pixel code  $P_i$  is repeated  $N$  times, where  $N$  is the Pixel Repeat Factor, and then the next pixel code  $P_{i+1}$  is read from the memory. As an example, for  $N = 2$ ,

Figure V.72 Example of Pixel Repeat Function



Pixel Repeat causes an image magnification effect in the horizontal direction. It is defined only for RGB555 and CLUT image coding types and not for DYUV and RL.

For CLUT 4, the unit that is repeated is the pixel pair.

The Pixel Repeat Factor is loaded via the DCP instruction 'Load display parameters' (V.4.6.1). In the Base Case it is limited to the numbers 2, 4, 8 and 16 only.

Note that when Pixel Repeat is applied to a display line, the default starting point for the next line is, as usual, the pixel immediately following the last **displayed** pixel on that line.

### 5.11.1.3 Line Hold / Repeat

In the vertical direction, mosaic effects are produced by repeating lines using the Display Line Start Pointers in the DCP. Line Repeat is produced by repeating line  $L_j$  by a factor  $M$ , then displaying line  $L_{j+1}$   $M$  times and so on. Line Hold is effected by repeating  $L_j$   $M$  times, then displaying line  $L_{j+M}$   $M$  times, etc.

Vertical repeat factors are of course independent of horizontal repeat factors, and may take any value. Under application control, they may also vary as a function of vertical position.

### 5.11.2 Effects

Among the effects that can be produced by the application using the mosaic facilities described above are granulation, magnification or zoom, and low resolution images.

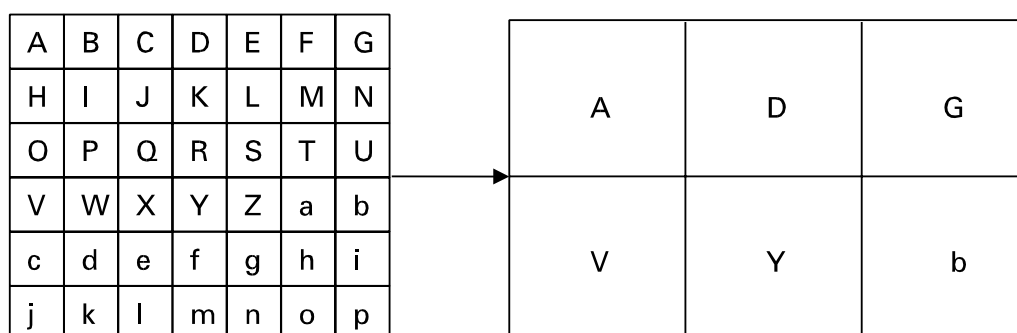
#### 5.11.2.1 Granulation

Granulation is an effect whereby the resolution of the image changes without its size altering. It is produced by a combination of Pixel Hold and Line Hold functions.

A typical use of granulation is to mask the loading of a new image from disc. While the image is loading, the current image is progressively granulated by increasing the hold factors from 1 upwards. A 'Cut' can then be performed to the new image, which is initially displayed with high factors and is progressively 'de-granulated' by reducing the factors to 1.

The following example of granulation uses pixel and line hold values of 3.

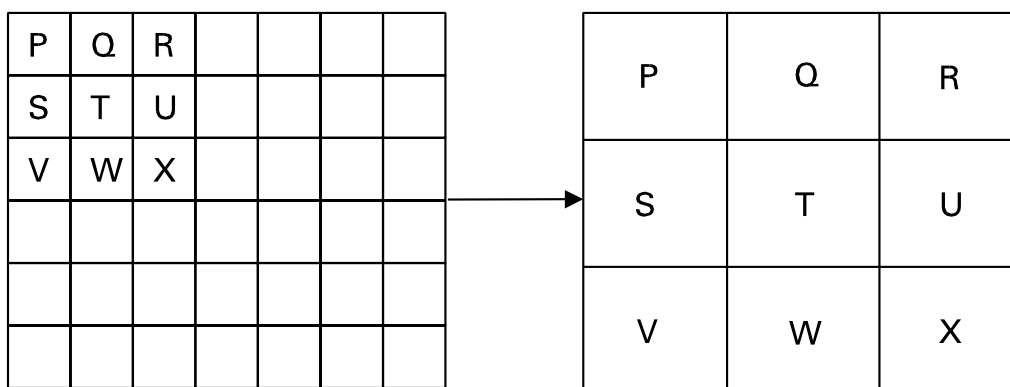
Figure V.73 Example of Granulation



#### 5.11.2.2 Magnification / Zoom

The pixel and line repeat functions may be used to produce an image magnification effect, e.g. for examining details of the structure of the image. This is restricted to RGB555 and CLUT images.

An example is shown in Figure V.74 below, with horizontal and vertical repeat factors of 2.

Figure V.74 **Example of Image Magnification Effect using Pixel and Line Repeat**

Note that the displayed image will be cut off at the right and bottom edges of the full (or reduced) screen. This may result in magnified pixels not being completely displayed.

### 5.11.2.3 Reduced Resolution

The magnification effect produced by pixel and line repeat may be used to produce a low resolution image, allowing fast update from disc of a larger than normal screen area. This is an alternative to animation by run-length compression, which may be chosen for specific applications. Techniques to improve the quality of a reduced resolution image depend on the implementation.



### 5.12 Cursor

The cursor plane is closest to the viewer, and is of dimensions 16 x 16 pixels.

The cursor plane may be enabled or disabled. It is a single color. Pixels in the cursor plane are one bit deep. If a pixel is set to 0 it is transparent and if it is set to 1 it appears in the programmed color. The cursor may be made to blink with programmable on and off periods. The blink type may be 'on/off' or 'normal color/complementary color'. The cursor color is defined by the intensity, red, green and blue components. For the Base Case, one bit per component is available giving the colors specified in V.5.13.

The cursor plane may be positioned at any coordinate over the other planes. The default cursor origin is always the top-left hand corner of the full-screen image. This is true whether the 'display compatibility mode' is zero or one. See V.4.8.

The cursor origin may be altered at any time by the application to be any position on or off of the display

The cursor co-ordinates are specified in UCM in High Resolution (see VII.2.3.4.2). However, in the Base Case, the positional accuracy of the cursor is only to Double Resolution. The resolution of the cursor bitmap itself may be set to Normal or Double Resolution for the Base Case System and also High Resolution for an Extended System.

The size of the cursor plane is limited. However, if a specific application requires a larger cursor, it may generate one itself in an image plane by software techniques. The cursor control commands are specified in VII.2.3.

V.5 Visual Effects

---

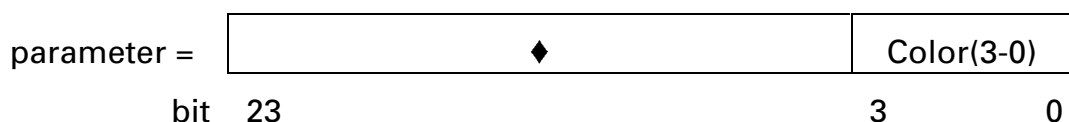
5.13 Backdrop

The backdrop is visible wherever all the enabled image planes and the cursor are transparent. The backdrop may be chosen to be a constant color (per line and per display path), or (in the extended case) to be synchronized external video (see V.4.6.1 for selection of external video).

The backdrop Color may be set by means of the Display Control Program instruction given below.

**Load Backdrop Color:**

Figure V.75 Load Backdrop Color Instruction



where

Color = Backdrop color.

The RGB components of the backdrop color, as a fraction of full intensity, are given by

$$\begin{aligned}
 B &= \text{Color (0)} * K \\
 G &= \text{Color (1)} * K \\
 R &= \text{Color (2)} * K
 \end{aligned}$$

Where  $K = 0.5$  when Color (3) = 0 and  $K = 1$  when Color (3) = 1.

Thus, in descriptive terms, the backdrop colors are as follows:

**Backdrop Color** (Binary values)

0000	Black	1000	Black
0001	Half-brightness Blue	1001	Blue
0010	Half-brightness Green	1010	Green
0011	Half-brightness Cyan	1011	Cyan
0100	Half-brightness Red	1100	Red
0101	Half-brightness Magenta	1101	Magenta
0110	Half-brightness Yellow	1110	Yellow
0111	Half-brightness White	1111	White

#### 5.14 Interlace

Display interlace may be turned on or off, by use of the UCM function DC\_Intl.

Whether the information is displayed on odd or even fields is the same or different depends on whether a normal/double resolution or a high resolution FCT/LCT pair are executed (by DC\_Exec).

For normal and double resolution images, a normal/double resolution FCT/LCT is used. Interlace may be turned on or off, with the normal recommended state being 'off'. If interlace is turned on, an interlaced normal or double resolution display will result, with the same information displayed on odd and even fields.

For high resolution images, a high resolution FCT/LCT pair is used, and interlace must be turned on.

Another combination is also possible, that is the use of a high resolution FCT/LCT with a normal or double resolution image coding method. By this means, the vertical resolution of such images may be doubled. In the case of double resolution (CLUT4 and RL3) the result is high resolution.

### V.6 On-disc Coding Structure and Data Formats

---

#### V.6 On-disc Coding Formats

##### 6.1 General

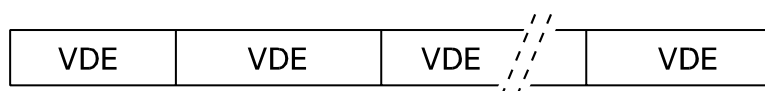
This section defines the on-disc coding structure for visual images. Video data can be contained in real time records together with control data (including DCP instructions) and, if appropriate, audio data. It can also be contained in non-real time files and loaded independently by the application.

## 6.2 Video Data Sequences

A video data sequence (VDSQ) is the basic unit of image data on the disc. It typically contains the pixel data for an image or a Partial Update to an image. A VDSQ is loaded to a drawmap.

Each video data sequence consists of a packed sequence of video data elements (VDEs).

Figure V.76 Video Data Sequence



A VDE typically represents a pixel, or some subdivision or indivisible multiple of a pixel. Elements within a segment are all of the same structure, as identified by the Video Coding Information Byte of each sector (see V.6.3).

### 6.2.1 Packing of VDSQ's

The Video Data Elements comprising a VDSQ are packed into (the data part of) one or more sectors with no gaps between elements. However, Video Data Sequences are not packed<sup>9</sup>. Each VDSQ must start at the beginning of a sector, and is contained in an integral number of sectors.

---

<sup>9</sup> Where this would use space on the disc inefficiently, i.e. for small VDSQ's it is recommended that a number of VDSQ's are combined and loaded as one larger VDSQ, and are separated later.

For example, a number of successive images for runlength animation can be loaded in one VDSQ into a single drawmap. They can then be displayed in sequence by loading their start addresses into the DCP at the right moment. Or a number of small Partial Updates can be loaded as a larger image and then copied one by one to their destination.

### 6.3 Video Coding Information Byte

#### 6.3.1 Format

Details of the coding method and resolution of each video data sequence are contained in the Video Coding Information Byte in the subheader of each sector of the sequence. The format of this byte is given below.

Figure V.77 **Format Video Coding Information Byte**

ASCF	EOLF	Resolution	Coding
------	------	------------	--------

bit 7                  6                  5                  4 3                  2                  1                  0

where

**ASCF = Application Specific Coding Flag**

- 0 The coding conforms to the specification of this chapter
- 1 The coding is Application-Specific i.e. to be interpreted by application software

**EOLF = Even/Odd Lines Flag**

- 0 Even
- 1 Odd

If error concealment is to be used, this flag indicates whether the sector contains even or odd lines of the image. If error concealment is not to be used, this bit is set to zero.

**Resolution** (Binary values)

- 00 Normal
- 01 Double
- 10 Reserved
- 11 High

**Coding** (Binary values)

0000	CLUT4	1000	QHY
0001	CLUT7	1001	Reserved
0010	CLUT8	1010	Reserved
0011	RL3	1011	Reserved
0100	RL7	1100	Reserved
0101	DYUV	1101	Reserved
0110	RGB555 (lower)	1110	Reserved
0111	RGB555 (upper)	1111	Reserved

## V.6 On-disc Coding Structure and Data Formats

---

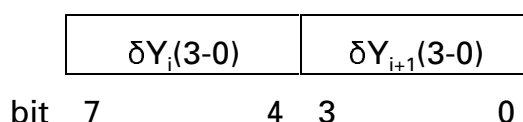
### 6.3.2 Example of Application Specific Coding

As an example of the use of the ASCF (Application Specific Coding Flag) in the Video Coding Information Byte, the Content Developer can code monochrome natural images on disc as follows.

Monochrome natural images may be encoded using the DYUV encoding technique, but with a Y component only ( $U = V = 0$ ). The U and V values need not be transmitted, thus the data on disc need have only 4 bits/pixel rather than 8. The storage space on disc is thus halved.

The on-disc format can be as follows:

Figure V.78 **Video Data Element**



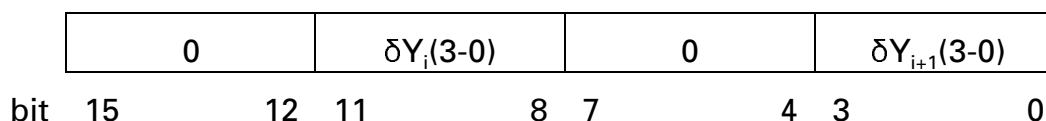
where  $i$  must be even and  $\delta Y = 4$  bit luminance difference code.

The VDE represents a pixel pair starting at an even X coordinate.

The ASCF bit for all such video sectors is set to 1, the EOLF bit and Resolution bits may be used according to V.6.3.1 as necessary and the Coding nibble (half-byte) may be arbitrarily set according to the meaning associated to it by the Content Developer.

In the decoder, the image data is loaded to a buffer (this can be a secondary drawmap) and each VDE is expanded in real-time by application software to the form below in image memory.

Figure V.79 **Video Data Element after expansion**



This expansion can conveniently be done by a table look-up method, with one 16 bit entry for each of the 256 possible VDEs.

## 6.4 Image Formats

### 6.4.1 General

In this section, the term "image" is used to mean an entire image, a Rectangular Partial Update or an Irregular Partial Update.

In general, the video data elements of an image are identical to the pixel codes in decoder memory, as given in V.4.4.

For all image coding methods except RGB555, an image is represented on disc by a single VDSQ.

The order of elements in a VDSQ representing an image follows the normal physical display scanning sequence, i.e. the elements are ordered in rows from top to bottom of the image, and within each row are ordered from left to right. Each row must consist of an integral number of longwords except for run length coded rows which must only be an integral number of bytes. Rows follow each other with no gaps or padding between, in all formats.

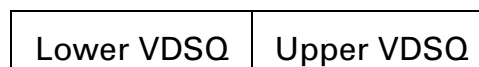
For information on image sizes when coding to 525 line, 625 line and 525/625 compatible formats, see V.3.2.



### 6.4.2 RGB555 Images

For RGB555, the data for an image is loaded in two parts, each corresponding to one of the two Image Stores. The two parts are formed from the lower (least significant) halves and upper (most significant) halves of each pixel code.

Figure V.80 **Example of RGB555 Image Format**



The two VDSQ's may be in either order and must each begin at the start of a new sector.

## V.6 On-disc Coding Structure and Data Formats

---

### 6.4.3 Error Concealment

In order to support error concealment by line repeat, as described in V.4.7, an image may be coded in two parts, one consisting of the even lines of the image, and the other the odd lines. Each part is made to occupy an integral number of sectors by, if necessary, filling the unused bytes of its final sector with zeros. The sectors of the two parts are then interleaved on disc to form the VDSQ for the image.

Adjacent odd and even lines are physically separated on disc by spacing the  $i^{\text{th}}$  odd sector from the  $i^{\text{th}}$  even sector by  $n$  physical sectors, where

$$2 \leq n \leq n_{\max}(r)$$

$n_{\max}$  is a linear function of radial position on the disc. For the inner track it is 7 and for the outer track it is 18.

This distance is chosen to minimise the chance that an even/odd pair of lines each have an error from the same physical disturbance (e.g. a hard error, a fingerprint, a scratch or dirt) on the disc.

The Grouping Factor for even and odd video sectors must be fixed for one VDSQ, i.e. if the Grouping Factor for a VDSQ is  $m$ , the even and odd sectors must be in alternate groups of  $m$  in the sector stream after removing any audio, data or empty sectors in the stream.

In the case of RGB555 coding, the image is split into four parts (lower even, upper even, lower odd, upper odd). The lower parts and upper parts are separately grouped as defined above.

As an example, if there is no file interleaving or audio sectors, a possible layout is:

Sector:	E1	E2	E3	O1	O2	O3	E4	E5	E6	O4	O5	O6	E7	E8	E9
Even	_____			_____			_____			_____			_____		
Odd	_____			_____			_____			_____			_____		

The even and odd sectors are directed to two separate memory areas in the decoder by placing their destination addresses in the Play Control Lists (see VII.2.2.3.2, SS\_Play). For the above example, where the Grouping Factor is 3, the Play Control Lists must be set up so that sectors E1 to E3 are directed to the PCL buffer 1, sectors O1 to O3 to PCL buffer 2 etc.

The decoder can then make use of e.g. line repeat, as described in V.4.7, to conceal lines found to be in error.

\*\*\*\*\* EXTENSION \*\*\*\*\*

#### 6.4.4 High Resolution Images

High Resolution images may be displayed either interlaced or line-sequential. The image on disc should comprise the even lines followed by the odd lines. The organization is identical to that used to support error concealment.

##### 6.4.4.1 DYUV + QHY

For High Resolution images coded according the QHY method of V.3.5, the image on disc consists of the Normal Resolution component accompanied by the QHY component.

Figure V.81 High Resolution Image on Disc



Note that the QHY component is divided into two parts, one for the even and the other for the odd lines.

##### 6.4.4.2 Compatibility with the Base Case

In the Extended Case, it is the responsibility of the application to ensure backward compatibility with the Base Case. In the case of High Resolution images, this may be achieved in many cases by accompanying the High Resolution image on disc, independent of the coding used, with a suitable image of Base Case, i.e. Normal or Double, resolution (see Chapter V.4.4.9).

The application, when running on a Base Case player, can load the Base Case resolution image instead of the High Resolution image.

For DYUV + QHY images, the application, when running on a Base Case player, can discard the QHY image and load only the DYUV image. This will achieve compatibility for the case of a natural image. If information vital to the application is lost, it is the responsibility of the application to present this information in some other form on a Base Case player.

\*\*\*\*\*

## 6.5 Pixel Data Formats

The disc formats of all the defined coding methods for pixel data are given in this section.

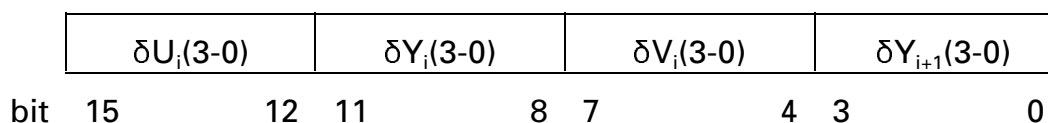
The index 'i' represents the pixel number along a horizontal line, starting at zero (except for runlength coding, where it represents run number).

### 6.5.1 Natural Images

#### 6.5.1.1 DYUV

The defined resolutions are Normal (**Base Case**) and High (**Extended Case**).

Figure V.82 Video Data Element DYUV



where i must be even,

$\delta Y$  = 4 bit luminance difference code, and  
 $\delta U, \delta V$  = 4 bit chrominance difference codes.

The VDE represents a pixel pair starting at an even X coordinate.

V.6 On-disc Coding Structure and Data Formats

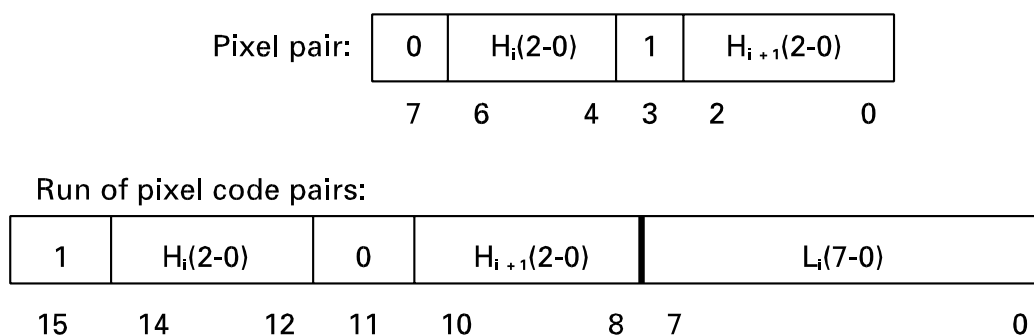
---

\*\*\*\*\* EXTENSION \*\*\*\*\*

6.5.1.2 QHY

The only defined resolution is High. The Video Data Elements are of variable length.

Figure V.83 Video Data Elements QHY



H = 3 bit QHY code.

L = 8 bit run length = Number of QHY codes in run

where  $2 \leq L \leq 255$  and

L = 1 is forbidden.

L = 0 means 'Continue this run to the end of the line'.

All lines must finish with this code for L, i.e a zero runlength.

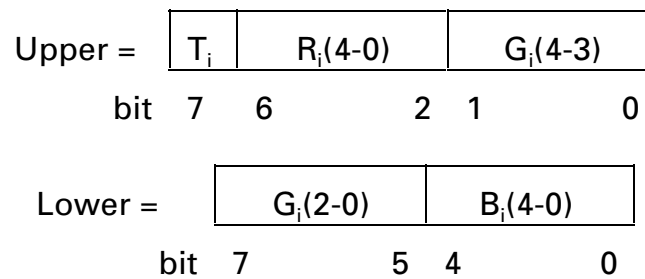
\*\*\*\*\*

**6.5.2 RGB555**

The only resolution available is Normal Resolution.

There are two types of VDE's and VDSQ's, upper and lower (see V.6.4.2).

**Figure V.84 Video Data Elements RGB555**



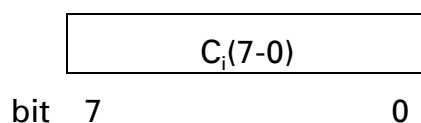
R,G,B = 5 bit Red, Green and Blue absolute values.  
 T = Transparency bit.

### 6.5.3 Color Lookup Table

#### 6.5.3.1 8-bit CLUT

The defined resolutions are Normal (**Base Case**) and High (**Extended Case**).

Figure V.85 Video Data Element 8-bit CLUT

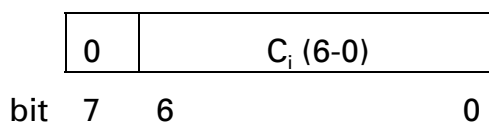


C = 8-bit CLUT address.

#### 6.5.3.2 7-bit CLUT

The defined resolutions are Normal (**Base Case**) and High (**Extended Case**).

Figure V.86 Video Data Element 7-bit CLUT

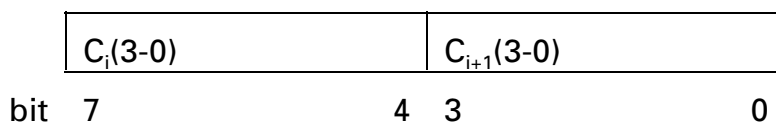


C = 7-bit CLUT address

#### 6.5.3.3 4-bit CLUT

The defined resolutions are Double (**Base Case**) and High (**Base Case in Interlace Mode**).

Figure V.87 Video Data Element 4-bit CLUT



C = 4-bit CLUT address.

where i must be even. The VDE represents a pixel pair starting at an even X coordinate.

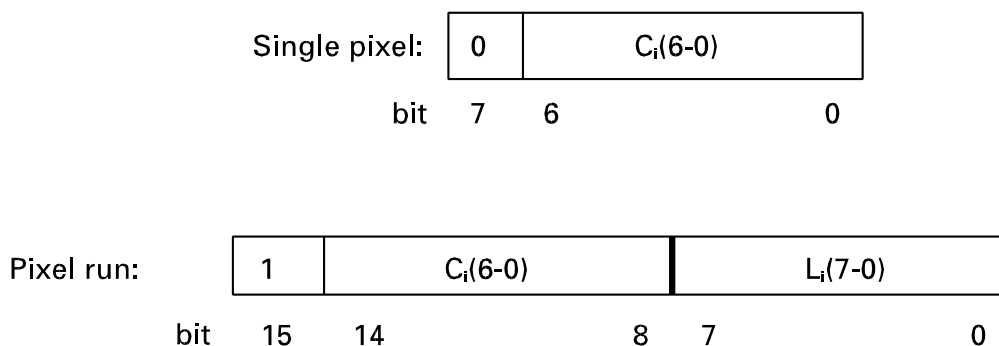
### 6.5.4 Run Length Encoded Images

#### 6.5.4.1 Run-coded 7-bit CLUT

The defined resolutions are Normal (**Base Case**) and High (**Extended Case**).

The Video Data Elements are of variable length.

Figure V.88 Video Data Elements RL 7-bit CLUT



C = 7-bit CLUT address.

L = 8 bit run length = Number of pixels in run

where  $2 \leq L \leq 255$  and

L = 1 is forbidden.

L = 0 means 'Continue this run to the end of the line'.

All lines must finish with this code for L, i.e a zero runlength.

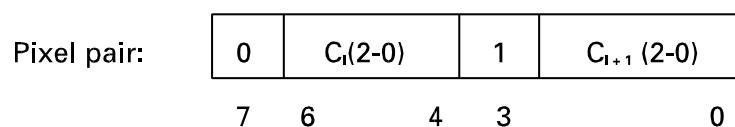


6.5.4.2 Run-coded 3-bit CLUT

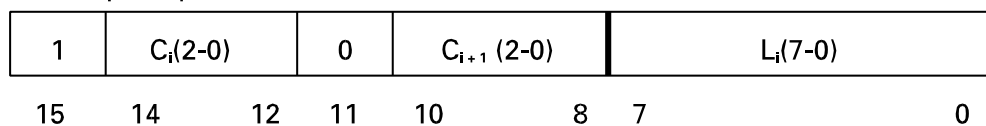
The defined resolutions are Double (**Base Case**) and High (**Base Case in Interlace Mode**).

The Video Data Elements are of variable length.

Figure V.89 Video Data Elements RL 3-bit CLUT



Run of pixel pairs:



$i$  = even

$C$  = 3 bit CLUT address.

$L$  = 8 bit run length = Number of pixel pairs in run

where  $2 \leq L \leq 255$  and

$L = 1$  is forbidden.

$L = 0$  means 'Continue this run to the end of the line'.

All lines must finish with this code for  $L$ , i.e a zero runlength.

This page is intentionally left blank

**Chapter VI Program-related Data Representations**

	<b>Page</b>
VI.1 General	VI-1
VI.2 Executable Object Code	VI-2
2.1 Object Code Files	VI-2
2.2 Instruction Set Summary	VI-3
2.3 System Modules	VI-5
VI.3 Data to be Processed by an Application	VI-6
3.1 Character Sets	VI-7
3.1.1 CD-I Default Character Set	VI-7
3.1.2 Other Character Sets	VI-9
3.2 Phonetic Coding	VI-10
3.3 Binary Data	VI-11
VI.4 Subheader Values	VI-12
4.1 General	VI-12
4.2 Coding Information Byte	VI-12

This page is intentionally left blank

## CD-I Full Functional Specification

Chapter VI

Program-related Data Representations

### List of Illustrations

---

<b>Fig. no.</b>	<b>Caption</b>	<b>Page</b>
VI.1	Summary of the 68000 MPU User State Instruction Set	VI-4
VI.2	Privileged 68000 Instructions for System State Routines	VI-5
VI.3	CD-I Default Character Set as given by ISO 8859-1	VI-8
VI-4	Coding Information Byte	VI-12

This page is intentionally left blank

VI.1 General

---

**VI.1 General**

The program related data is the data to be read and processed by a microprocessor of the 68000 MPU family. It is stored in 2048 byte Data blocks and should not be mixed in the same sector with audio or video data. These Data blocks are stored in Mode 2, Form 1 sectors, with the submode byte data bit set to one.

Different types of data present on a CD-I disc are distinguished as executable object code and data to be processed by the application.

CD-RTOS does not restrict the format or contents of data to be processed by an application program. This data can be:

- Text
- Phonetic coding
- Binary data

## VI.2 Executable Object Code

---

### VI.2 Executable Object Code

#### 2.1 Object Code Files

Object code is always stored in separate 'object code' files. Such an object code file is stored in Mode 2, Form 1 sectors and does not contain audio or video sectors. The submode byte of each sector of an object code file must have the data bit set to one. In addition, the last sector of the file must have the EOF bit set to one. The coding information byte for object code files is undefined and must be set to zero.

The unit of a loadable program is one named file, which may contain one or several software modules. These files may be located anywhere within the CD-I file system. By convention, they are usually placed in a directory named 'CMDS'.

Software modules are usually created by the CD-RTOS linker. A complete definition of the module format is given in Appendix VII.1 (chapter 1).

Object code files are not real-time files. If a data error occurs that cannot be corrected by the CD-I controller hardware or device driver, the affected module(s) may not be loaded or executed.



VI.2 Executable Object Code

---

**2.2 Instruction Set Summary**

The 68000 MPU family includes several similar, but not identical, microprocessors. To be compatible with all players, the object code on a CD-I disc must be written for the microprocessor with features which are common to all (e.g., to the 68000 itself). This instruction set is defined in the 'M68000 16/32-Bit Microprocessor Programmer's Reference Manual', published by Motorola (document number M68000UM).

One incompatibility exists in the 68000 family. The 68000 MPU instruction set allows a 'move from sr' instruction to be executed by user state programs. Other processors regard this as a privilege violation and provide the 'move from ccr' instruction instead. Unfortunately, 'move from ccr' is not a legal 68000 instruction.

To overcome this difference, application programs may not use the 'move from sr' instruction. Instead, the 'move from ccr' instruction must be used. This instruction is emulated by the CD-RTOS kernel on players which use the 68000 (or other) MPU. Since emulation is slow, it is recommended that the 'scc' instruction be used to save the state of the condition code register.

CD-RTOS provides a service request (F\$SetSys) to examine system global variables. One of these variables (D\_MPUTyp) is initialized by the system to contain the MPU actually present on a particular CD-I player. Programs wishing to exploit extended MPU features may inspect this variable and use processor specific routines to take advantage of the native instruction set. However, it is always mandatory to provide functionally equivalent routines which use only the instructions and addressing modes available on a 68000 MPU. The application must take care of the compatibility problems resulting from the difference in performance when executed on the Base Case System.

The 68000 MPU user state instruction set is summarized in Figure VI.1. The conditions that may be tested in the 'bcc', 'dbcc', and "scc" instructions in Figure VI.1 are:

cc	carry clear	lt	less than
cs	carry set	hi	higher
eq	equal	hs	higher or same
ge	greater or equal	ne	not equal
gt	greater than	mi	minus
lo	lower	pl	plus
le	less than or equal	vc	overflow clear
ls	lower or same	vs	overflow set

## VI.2 Executable Object Code

Figure VI.1 Summary of the 68000 MPU User State Instruction Set

Instructions	Description
abcd	add decimal
add, adda, addi, addq, addx	add binary
and, andi, andi to ccr	logical and
asl	arithmetic shift left
asr	arithmetic shift right
bcc	conditional branch
bchg	bit test and change
bclr	bit test and clear
bra	unconditional branch
bset	bit test and set
chk	check register against bounds
clr	clear to zero
cmp, cmpa, cmpi, cmpm	compare
dbcc, dbra	test, decrement, and branch
divs, divu	signed and unsigned divide
eor, eori, eori to ccr	logical exclusive or
exg	exchange registers
ext	sign extend
jmp	jump
jsr	jump to subroutine
lea	load effective address
link	link and allocate stack
lsl	logical shift left
lsr	logical shift right
move, movea, movem, movep, moveq, move to or from ccr	move data
muls, mulu	signed or unsigned multiply
nbcd	negate decimal
neg, negx	negate binary
nop	no operation
not	logical complement
or, ori, ori to ccr	logical inclusive or
pea	push effective address
rol, roxl	rotate left
ror, roxr	rotate right
rts, rtr	return from subroutine
sbcd	subtract decimal
scc	set according to condition
sub, suba, subi, subq, subx	subtract binary
swap	swap register halves
tas	test and set
trap, os9, tcall	call system or library routine
trapv	trap on overflow
tst	compare with zero
unlk	de allocate stack and unlink

## VI.2 Executable Object Code

## 2.3 System Modules

CD-RTOS (see VII and Appendix VII.1) has two distinct environments in which object code may be executed - **user** state, and **system** state. Application programs are executed in user state. File managers and device drivers are always executed in system state.

A system state routine has access to the entire capabilities of the hardware. For example, on memory protected systems, a system state routine may access any memory in the system. It may alter internal system data structures, mask interrupts, or take direct control of hardware interrupt vectors if necessary. There are also a small group of CD-RTOS service requests accessible only from system state.

System state is synonymous with the 68000 MPU family **supervisor** state. The 68000 MPU system state instruction set includes a few **privileged instructions** that may be used in system state routines. This instruction set is outlined in Figure VI.2.

Figure VI.2 Privileged 68000 Instructions for System State Routines

Instruction	Description
andi to sr	logical and to status register
eori to sr	exclusive or to status register
move to sr, move from sr	move word to/from status register
move usp	move to/from user stack pointer
ori to sr	inclusive or to status register
reset <sup>1</sup>	reset external devices
rte	restore status register and return
stop <sup>1</sup>	halt processor

---

<sup>1</sup> The reset and stop instructions are usually not used.

VI.2 Executable Object Code

---

**VI.3 Data to be Processed by an Application**

All program related data that is to be processed by an application is stored in Mode 2, Form 1 sectors on the disc. The data bit in the submode byte of each sector must be set to one.

Such data may be included in real-time records, typically at the beginning of a real-time record. If such data is not at or near the beginning of a real-time record it is **recommended** that they be placed before the real-time record as non real-time sectors.

## VI.2 Executable Object Code

---

### 3.1 Character Sets

Programs that display textual data are required to have a font module associated with the character set that is used. A font module is a standard CD-RTOS data module that contains the image of each character in the character set. The format of a font module is described further in VII.2.3.2.10.

#### 3.1.1 CD-I Standard Character Set

The CD-I standard character set conforms to the ISO 8859-1 character set. The font module for this character set must be in ROM in the Base Case System. A copy of this character set is shown in Figure VI.3.

The following characters have special meaning when used with this font module for display via UCM.

\$08 H	Cursor Left
\$0A H	Cursor Down
\$0B H	Cursor Up
\$0C H	Cursor Right
\$0D H	Carriage Return
\$1E H	Cursor Home

VI.2 Executable Object Code

Figure VI.3 CD-I default character set as given by ISO 8859-1

				b8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				b7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
				b6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				b5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b4	b3	b2	b1	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	
0	0	0	0	00		SP	0	@	P	`	p		NBSP	°	À	Ð	à	ð		
0	0	0	1	01		!	1	A	Q	a	q		ı	±	Á	Ñ	á	ñ		
0	0	1	0	02		"	2	B	R	b	r		¢	²	Â	Ò	â	ò		
0	0	1	1	03		#	3	C	S	c	s		£	³	Ã	Ó	ã	ó		
0	1	0	0	04		\$	4	D	T	d	t		¤	´	Ä	Ô	ä	ô		
0	1	0	1	05		%	5	E	U	e	u		¥	µ	Å	Õ	å	õ		
0	1	1	0	06		&	6	F	V	f	v			¶	Æ	Ö	æ	ö		
0	1	1	1	07		'	7	G	W	g	w		§	•	Ç	×	ç	÷		
1	0	0	0	08		(	8	H	X	h	x		¨	˘	È	Ø	è	ø		
1	0	0	1	09		)	9	I	Y	i	y		©	¹	É	Ù	é	ù		
1	0	1	0	10		*	:	J	Z	j	z		ª	º	Ê	Ú	ê	ú		
1	0	1	1	11		+	;	K	[	k	{		«	»	Ë	Û	ë	û		
1	1	0	0	12		,	<	L	\	l			¬	¼	Ì	Ü	ì	ü		
1	1	0	1	13		-	=	M	]	m	}		SHY	½	Í	Ý	í	ý		
1	1	1	0	14		.	>	N	^	n	~		®	¾	Î	Þ	î	þ		
1	1	1	1	15		/	?	O	_	o			-	¿	Ï	ß	ï	ÿ		

## VI.2 Executable Object Code

---

### 3.1.2 Other Character Sets

A coded character set can be any character set identified in the ISO International Register of Coded Character Sets to be Used with escape sequences or shifted JIS Kanji. To display characters for a coded character set, the font module must reside in memory before displaying the characters. A font module can be stored in ROM or on the disc. Because a font module (i.e. other than for the standard character set) may not be in ROM on all systems, any font module other than for ISO 8859-1 must reside on each disc where the font is used. Functions are supplied in UCM for loading and accessing font modules.

When textual information that is used by CD-RTOS is stored in files, the text can be encoded in any coded character set. If a character set other than ISO 8859-1 is used, the corresponding font module must be installed before any system function that uses the font module is executed. Font modules can also be used for representing images other than characters.

It should be noted that it is the responsibility of the application to ensure that characters are both visible and legible on the display screen. As for the number of characters (e.g. 40 or 80) which can be displayed across the Safety Area of the screen, the application needs to access and use the information contained in the System's Configuration Status Descriptor (CSD, see A VII.2.4) in order to assure legibility.

### 3.2 Phonetic Coding

Text that represents phonetic coding to be used for speech may be coded on the disc using the ISO 8859-1 character set. This text is read by an application program which converts it into 8-bit PCM data (i.e., filter gain values K0 and K1 equal to zero). This data is then output via CDFM functions described in VII.2.2.4. The format of the PCM data is described in Chapter IV.



### 3.3 Binary Data

Binary data is data that is to be decoded by an application program. CD-RTOS places no restrictions on the format or content of binary data.

VI.4 Subheader values

---

**VI.4 Subheader Values****4.1. General**

In General, the subheader values for all data sectors discussed in this chapter are:

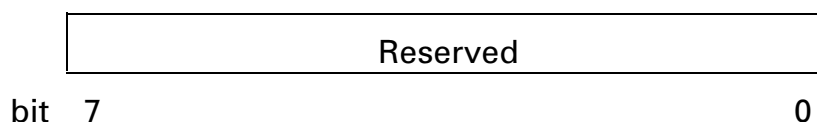
File number:	% xxxxxxxx
Channel number:	% 000xxxxx
Submode:	% x00x100x

e.g. executable object code is non-real time, Mode 2, Form 1 with the data bit set and audio/video bits having the value zero.

**4.2. Coding Information Byte**

The data or program related data Coding Information byte is defined as follows:

Figure VI.4 **Coding Information Byte**



**Chapter VII - Compact Disc Real Time Operating System (CD-RTOS)**

	<b>Page</b>
VII.1 Kernel	VII-1
1.1 General	VII-1
1.1.1 CD-RTOS User State Service Request Summary	VII-1
1.1.2 CD-RTOS I/O Service Request Summary	VII-3
1.1.3 CD-RTOS System State Service Request Summary	VII-4
1.1.4 Signal Handling Facilities	VII-4
1.2 CD-RTOS Startup Procedures	VII-6
Manufacturer dependent routines	VII-6
CD-RTOS routines	VII-6
Initial process routines	VII-7
1.3 Hardware Configuration Status Descriptor	VII-9
1.3.1 General	VII-9
1.3.2 Compilation of the CSD	VII-9
1.4 Application Startup Environment	VII-10

This page is intentionally left blank

## Table of Contents

		<b>Page</b>
VII.2	File Managers	VII-11
2.1	General	VII-11
2.2	Compact Disc File Manager (CDFM)	VII-14
2.2.1	Introduction	VII-14
2.2.2	Features and Concepts	VII-15
2.2.2.1	Execution Classes	VII-15
2.2.2.2	Scheduling	VII-15
2.2.2.3	File Security System	VII-16
2.2.2.4	Audio Scheduling	VII-16
2.2.2.5	Directory Sector Buffering	VII-17
2.2.3	CD Service Requests	VII-18
2.2.3.1	I\$GetStt - Getstat Functions	VII-18
	SS_Opt	VII-19
	SS_EOF	VII-20
	SS_Size	VII-21
	SS_Pos	VII-22
	SS_CDFD	VII-23
	SS_Path	VII-24
2.2.3.2	I\$SetStt - Setstat Functions	VII-25
	SS_Seek	VII-26
	SS_Abort	VII-28
	SS_CDDA	VII-29
	SS_Pause	VII-30
	SS_Eject	VII-31
	SS_Mount	VII-32
	SS_Opt	VII-33
	SS_Disable	VII-34
	SS_Enable	VII-35
	SS_Cont	VII-36
	SS_Stop	VII-37
	SS_Raw	VII-38
	SS_ReadTOC	VII-39
	SS_Play	VII-42
	Play Control Block	VII-44
	Play Control List	VII-48
	Play Termination	VII-52
	SS_CChan	VII-53

This page is intentionally left blank

## Table of Contents

	<b>Page</b>	
2.2.3.3	I\$Seek	VII-54
2.2.3.4	I\$Read	VII-55
2.2.3.5	I\$ReadLn	VII-56
2.2.3.6	I\$ChgDir	VII-57
2.2.4	Audio Functions	VII-58
2.2.4.1	Soundmap Control Functions	VII-58
	SM_Creat	VII-58
	SM_Out	VII-60
	SM_Off	VII-61
	SM_Close	VII-62
	SM_Cncl	VII-63
	SM_Info	VII-64
	SM_Stat	VII-65
2.2.4.2	Sound Data Manipulation Functions	VII-66
	SD_MMix	VII-66
	SD_SMix	VII-67
	SD_Loop	VII-68
2.2.4.3	Sound Control Functions	VII-69
	SC_Atten	VII-69
2.2.5	Open and Close Service Requests	VII-70
2.2.5.1	I\$Open	VII-70
2.2.5.2	I\$Close	VII-72
2.3	User Communications Manager	VII-73
2.3.1	Introduction	VII-73
2.3.2	Features and Concepts	VII-74
2.3.2.1	Environment	VII-74
2.3.2.2	Display Architecture	VII-75
2.3.2.3	Drawmaps	VII-76
2.3.2.4	Display Control Program	VII-77
	Video events	VII-79
	High resolution	VII-79
	DCP instructions	VII-79

This page is intentionally left blank



## Table of Contents

	<b>Page</b>	
2.3.2.5	Coordinate Transformation	VII-83
2.3.2.6	Regions	VII-83
2.3.2.7	Graphics Drawing	VII-85
2.3.2.8	Pattern Indirect Drawing	VII-85
2.3.2.9	Character Output Encoding	VII-85
2.3.2.10	Text Fonts	VII-86
2.3.3	General UCM Service Requests	VII-89
2.3.3.1	I\$Open	VII-89
2.3.3.2	I\$Close	VII-90
2.3.3.3	SS_SLink	VII-90
2.3.4	Video and Graphics Functions	VII-92
2.3.4.1	Drawmap Control Functions	VII-92
	DM_Creat	VII-93
	Drawmap Descriptor Format	VII-95
	Drawmap Data Type Format	VII-97
	Resolution/Data Type	VII-97
	Combinations/Restrictions	
	DM_Org	VII-98
	DM_Copy	VII-99
	DM_Exch	VII-100
	DM_TCpy	VII-101
	DM_TExc	VII-102
	DM_Write	VII-103
	DM_IrWr	VII-104
	DM_Read	VII-105
	DM_WrPix	VII-106
	DM_RdPix	VII-107
	DM_Cncl	VII-108
	DM_Close	VII-109
	DM_DMDup	VII-110
2.3.4.2	Graphics Cursor Functions	VII-111
	GC_Pos	VII-111
	GC_Show	VII-112
	GC_Hide	VII-113
	GC_Ptn	VII-114
	GC_Col	VII-115
	GC_Org	VII-116
	GC_Blnk	VII-117

This page is intentionally left blank

## Table of Contents

	<b>Page</b>
2.3.4.3	Regions VII-118
	RG_Creat VII-119
	RG_Isect VII-121
	RG_Union VII-122
	RG_Diff VII-123
	RG_XOR VII-124
	RG_Move VII-125
	RG_Del VII-126
2.3.4.4	Drawing Parameter Functions VII-127
	DP_Ptn VII-127
	DP_PAln VII-128
	DP_SCMM VII-129
	DP_SCR VII-130
	DP_GFnt VII-131
	DP_AFnt VII-132
	DP_Dfnt VII-133
	DP_RFnt VII-134
	DP_Clip VII-135
	DP_PnSz VII-136
	DP_PStyl VII-137
	DP_TCol VII-138
2.3.4.5	Graphics Drawing Functions VII-139
	DR_Dot VII-141
	DR_Line VII-142
	DR_PLin VII-143
	DR_CArc VII-144
	DR_EArc VII-145
	DR_Rect VII-146
	DR_ERect VII-147
	DR_PGon VII-148
	DR_Circ VII-149
	DR_CWdg VII-150
	DR_Elps VII-151
	DR_EWdg VII-152
	DR_DRgn VII-153
	DR_BFil VII-154
	DR_FFil VII-155
	DR_Copy VII-156
	DR_Text VII-157
	DR_JTxt VII-158

This page is intentionally left blank

## Table of Contents

	<b>Page</b>	
2.3.4.6	Display Control Functions	VII-159
	DC_CrFCT	VII-159
	DC_RdFCT	VII-160
	DC_WrFCT	VII-161
	DC_RdFI	VII-162
	DC_WrFI	VII-163
	DC_DIFCT	VII-164
	DC_CrLCT	VII-165
	DC_RdLI	VII-166
	DC_WrLI	VII-167
	DC_DILCT	VII-168
	DC_RdLCT	VII-169
	DC_WrLCT	VII-170
	DC_PRdLCT	VII-171
	DC_PWrLCT	VII-172
	DC_NOP	VII-173
	DC_SSig	VII-174
	DC_Relea	VII-175
	DC_SetCmp	VII-176
	DC_Intl	VII-177
	DC_FLnk	VII-178
	DC_LLnk	VII-179
	DC_Exec	VII-180
	DC_SetAR	VII-180
2.3.4.7	Video Inquiry Functions	VII-181
	VIQ_TxtL	VII-181
	VIQ_CPos	VII-182
	VIQ_JCPs	VII-183
	VIQ_FDta	VII-184
	VIQ_GDta	VII-185
	VIQ_RGInfo	VII-186
	VIQ_PntR	VII-187
	VIQ_RLoc	VII-188
	VIQ_DMInfo	VII-189
2.3.4.8	Character Output Functions	VII-190
	I\$WriteLn	VII-190
	I\$Write	VII-191
	CO_COD	VII-192
	CO_SCMM	VII-193
	CO_AFnt	VII-194
	CO_DFnt	VII-195

This page is intentionally left blank

## Table of Contents

	<b>Page</b>	
2.3.4.9	CRT Terminal Emulation Functions	VII-196
	Cursor Up	VII-196
	Cursor Down	VII-196
	Cursor Right	VII-196
	Cursor Left	VII-196
	Cursor Home	VII-197
	Carriage Return	VII-197
	Cursor Address	VII-197
	Delete Line	VII-197
	Insert Line	VII-197
	Show Cursor	VII-197
	Hide Cursor	VII-197
	Clear to End of Line	VII-197
	Clear to End of Screen	VII-197
	Start Reverse Video	VII-198
	End Reverse Video	VII-198
	Start Underlining	VII-198
	End Underlining	VII-198
	Clear Screen	VII-198
	Insert Character	VII-198
	Delete Character	VII-198
	Turn On Auto Wrap	VII-198
	Turn Off Auto Wrap	VII-199
2.3.5	Pointer Input Functions	VII-200
	PT_Coord	VII-200
	PT_SSig	VII-201
	PT_Relea	VII-202
	PT_Org	VII-203
	PT_Pos	VII-204
2.3.6	Keyboard Input Functions	VII-205
	I\$Read	VII-205
	KB_Rdy	VII-206
	KB_Read	VII-207
	KB_SSig	VII-208
	KB_Rel	VII-209
	KB_Repeat	VII-210
	KB_Stat	VII-211
2.3.7	Player Control Key Functions	VII-212
	KB_Avail	VII-212
	KB_NrAvail	VII-213

This page is intentionally left blank



Table of Contents

---

	<b>Page</b>
2.4 Non-volatile RAM File Manager (NRF)	VII-214
2.4.1 General	VII-214
2.4.2 I\$Create	VII-215
2.4.3 I\$Open	VII-216
2.4.4 I\$Delete	VII-217
2.4.5 I\$Seek	VII-218
2.4.6 I\$Read, I\$ReadLn	VII-219
2.4.7 I\$Write, I\$WriteLn	VII-220
2.4.8 I\$Close	VII-221
2.4.9 Status Functions	VII-222
2.4.10 Changing and Making Directories	VII-223
2.4.11 NRF Directories	VII-224
2.5 Error codes	VII-224

This page is intentionally left blank

## Table of Contents

		<b>Page</b>
VII.3	Device Drivers	VII-225
3.1	Compact Disc	VII-225
3.1.1	Introduction	VII-225
3.1.2	Basic Functional Requirements of CDFM Drivers	VII-225
3.1.3	Driver Module Format	VII-226
3.1.4	Data Structures	VII-227
3.1.4.1	CDFM Device Descriptors	VII-227
3.1.4.1.1	Standard Device Descriptor Header Fields	VII-228
3.1.4.1.2	Device Descriptor Option Fields	VII-228
3.1.4.1.3	Device Configuration Fields	VII-228
3.1.4.2	CDFM Path Descriptors	VII-229
3.1.4.2.1	Standard Path Descriptor Fields	VII-229
3.1.4.2.2	File Manager Path Descriptor Fields	VII-230
3.1.4.2.3	Option Table Path Descriptor	VII-231
3.1.4.3	CDFM Device Driver Static Storage	VII-232
3.1.4.3.1	Drive Tables	VII-233
3.1.5	CDFM Device Driver Subroutines	VII-235
3.1.5.1	The INIT Subroutine	VII-236
3.1.5.2	The READ Subroutine	VII-236
3.1.5.3	The WRITE Subroutine	VII-237
3.1.5.4	The GETSTAT and SETSTAT Subroutines	VII-237
	SS_Raw	VII-238
	SS_Read TOC	VII-238
	SS_Seek	VII-238
	SS_CDDA	VII-238
	SS_Mount	VII-238

This page is intentionally left blank

## Table of Contents

	<b>Page</b>	
3.1.5.5	The TERMINATE Subroutine	VII-239
3.1.5.6	The TRAP Subroutine	VII-240
3.1.5.7	The PLAY Subroutine	VII-241
3.1.5.8	The IRQ Service Request Subroutine	VII-242
<b>3.2</b>	<b>UCM Driver Interface</b>	<b>VII-243</b>
3.2.1	Driver and Descriptor Organization and Initialization	VII-243
3.2.2	Data Structures	VII-244
3.2.2.1	Device Descriptor Format	VII-244
3.2.2.1.1	Standard Device Descriptor Fields	VII-244
3.2.2.1.2	Device Descriptor Option Fields	VII-244
3.2.2.2	Path Descriptor Format	VII-245
3.2.2.2.1	Standard Fields	VII-245
3.2.2.2.2	File Manager Fields	VII-245
3.2.2.2.3	Options Fields	VII-247
3.2.2.3	Driver Static Storage Format	VII-248
3.2.3	Driver Functions	VII-249
3.2.3.1	Init	VII-249
3.2.3.2	Read	VII-250
3.2.3.3	Write	VII-251
3.2.3.4	GetStat and SetStat	VII-252
3.2.3.5	Term	VII-253
3.2.3.6	The TRAP Subroutine	VII-253
3.2.3.7	IRQ	VII-255
<b>3.3</b>	<b>NRF Driver Interface</b>	<b>VII-256</b>
3.3.1	General	VII-256
3.3.2	Device Descriptor Format	VII-257
3.3.2.1	Standard Device Descriptor Fields	VII-257
3.3.2.2	Device Descriptor Option Fields	VII-257

This page is intentionally left blank

Table of Contents

---

	<b>Page</b>
3.3.3 Path Descriptor Format	VII-258
3.3.3.1 Standard Fields	VII-258
3.3.3.2 File Manager Fields	VII-258
3.3.3.3 Option Fields	VII-258
3.3.4 Driver Static Storage	VII-259
3.3.4.1 Logical Format of NVRAM Area	VII-259
3.3.5 Driver Functions	VII-260
3.3.5.1 Init	VII-260
3.3.5.2 Read	VII-261
3.3.5.3 Write	VII-262
3.3.5.4 GetStat and SetStat	VII-263
3.3.5.5 Term	VII-264
3.3.5.6 The TRAP Subroutine	VII-265
VII.4 Synchronization Primitives	VII-266

This page is intentionally left blank



## List of Illustrations

<b>Fig. no.</b>	<b>Caption</b>	<b>Page</b>
VII.1	CD-RTOS User State Service Requests	VII-2/3
VII.2	CD-RTOS I/O Service Requests	VII-3
VII.3	CD-RTOS System State Service Requests	VII-4
VII.4a	Functions Provided by the File Managers	VII-12
VII.4b	Data Flow Model for CD-RTOS and its Relation to the Hardware	VII-12
VII.5	Subroutine Entry Points for the CD-File Manager	VII-14
VII.6	GetStat Service Requests in the CDFM	VII-18
VII.7	SetStat Service Requests in the CDFM	VII-25
VII.8	Structure of a Status Block	VII-26
VII.9	ASY_Stat Field	VII-26
VII.10	Table of Contents Structure Format	VII-40
VII.11	Table of Disc Types	VII-41
VII.12	Play Control Block Format	VII-44
VII.13	CIL for Audio data	VII-46
VII.14	CIL for Video/Program Related Data	VII-47
VII.15	Structure of a Play Control List	VII-48
VII.16	Error Information Structure	VII-50
VII.17	UCM Service Requests	VII-74
VII.18	Functional Organization of the User Communication Manager	VII-75
VII.19	Example of the Video Section of a CD-I Player	VII-75
VII.20	Example of a DCP and its Sequence of Interpretation	VII-78

This page is intentionally left blank

List of Illustrations

---

<b>Fig. no.</b>	<b>Caption</b>	<b>Page</b>
VII.21	Example of a Multiple LCT Display Control Program	VII-79
VII.22	Control Program Instructions for Path 0 only	VII-81
VII.23	Control Program Instructions for Path 1 only	VII-81
VII.24	Control Program Instructions for both Planes	VII-82
VII.25	Region Header format	VII-84
VII.26	Structure of Text Font Data Section	VII-87
VII.27	Structure of each Entry in the Glyph Data Table	VII-88
VII.28	Resolution/Data Type Combinations/ Restrictions	VII-97
VII.29	Status Functions supported by NRF	VII-222
VII.30	CDFM Device Descriptor Option Fields	VII-228
VII.31	Standard Path Descriptor Fields	VII-229
VII.32	CD File Manager Path Descriptor Fields	VII-230
VII.33	Option Table of a CDFM Path Descriptor	VII-231
VII.34	CDFM Device Driver Static Storage	VII-232
VII.35	CDFM Drive Table Format	VII-233
VII.36	V_PlayFlag Field	VII-234
VII.37	UCM Device Descriptor Option Fields	VII-244
VII.38	UCM File Manager Fields	VII-245
VII.39	UCM File Manager Video Driver Fields	VII-246
VII.40	UCM Options Fields of a Path Descriptor	VII-247
VII.41	Standard Driver Static Storage Fields	VII-248
VII.42	UCM Driver Static Storage Format	VII-248

This page is intentionally left blank

List of Illustrations

---

<b>Fig. no.</b>	<b>Caption</b>	<b>Page</b>
VII.43	Device Descriptor Option Fields	VII-257
VII.44	File Manager Fields	VII-258
VII.45	Option Fields	VII-258
VII.46	NVRAM Logical Addresses	VII-259
VII.47	NVRAM File Manager	VII-259

This page is intentionally left blank

**Chapter VII CD-RTOS (Compact Disc Real-time Operating System)****VII.1 Kernel****1.1 General**

All application programs issue requests for system services through the CD-RTOS kernel. For detailed information about the internal operations of the CD-RTOS kernel, see "The Kernel" part of A VII.1.

Three types of system services are provided by CD-RTOS. The 'user state' service requests provide application programs with basic functions such as memory allocation, process scheduling, and memory module management. The I/O service requests provide device independent input/output facilities. Device drivers and other operating system components may use the 'system state' requests to perform low-level functions, such as the installation of interrupt service routines, or new service requests.

The "OS-9 System Calls" part of A VII.1 contains detailed explanations of all the service requests available in CD-RTOS. A brief summary of the service requests is presented here for reference.

**1.1.1 CD-RTOS User State Service Request Summary**

All of the user state service requests available in CD-RTOS are outlined in Figure VII.1 below.

Figure VII.1 CD-RTOS User State Service Requests

Name	Description of Use
F\$Alarm	Send alarm signal after specified time period
F\$AllBit	Allocate bits in an allocation bitmap
F\$CCtl	Cache control
F\$Chain	Terminate the current process and begin a new one
F\$CmpNam	Determine if two strings match
F\$CpyMem	Copy a block of memory
F\$CRC	Calculate a 24-bit CRC over a block of memory
F\$DatMod	Create a named data module and return its address
F\$DelBit	Deallocate bits in an allocation bitmap
F\$DExec	Execute (n) instructions in a debugged program
F\$DExit	Terminate a debugged program
F\$DFork	Start a new process under control of the debugger
F\$Event	Create, manipulate and delete events
F\$Exit	Terminate the current process
F\$Fork	Start a new concurrent process
F\$GBlkMp	Return system free memory statistics
F\$GModDr	Return a copy of the system module directory
F\$GPrDBT	Return a copy of the system process table
F\$GPrDsc	Return a copy of a given process descriptor
F\$Gregor	Convert date and time to Gregorian format
F\$ID	Return the caller's process ID number
F\$Icpt	Install an intercept routine to process signals
F\$Julian	Convert date and time to Julian format
F\$Link	Return the address of a module in memory
F\$Load	Load memory module(s) from a file
F\$Mem	Change the caller's primary data area size
F\$PErr	Report a system error message
F\$PrsNam	Validate and parse a path name
F\$RTE	Return to main program after signal processing
F\$SchBit	Search for free bits in an allocation bitmap
F\$Send	Send a signal to a process
F\$SetCRC	Generate a valid CRC in a module in memory
F\$SetSys	Examine or change a system global variable
F\$SigMask	Enable or disable signal interrupts
F\$Sleep	Suspend the calling process for a period of time
F\$SPrior	Change a process's priority
F\$SRqCMem	Request memory by type
F\$SRqMem	Allocate a block of memory
F\$SRtMem	Deallocate a block of memory
F\$SSpd	Suspend process
F\$STime	Set the current date and time
F\$STrap	Install a routine to catch execution errors
F\$SUser	Change the caller's user ID number
F\$SysDbg	Invoke the system level debugger (if present)

(continued)



Figure VII.1 **CD-RTOS User State Service Requests** (continued)

Name	Description of Use
F\$Time	Return the current date and time
F\$Trans	Translate memory dates
F\$UAcct	User accounting
F\$TLink	Install a trap handling library module
F\$UnLink	Notify system that a memory module is not needed
F\$UnLoad	Same as F\$UnLink except by name instead of address
F\$Wait	Wait for forked process to terminate

### 1.1.2 CD-RTOS I/O Service Request Summary

All of the I/O service requests available in CD-RTOS are outlined in Figure VII.2 below.

Figure VII.2 **CD-RTOS I/O Service Requests**

Name	Description of Use
I\$Attach	Attach I/O device
I\$ChgDir	Change default directory
I\$Close	Close path
I\$Create	Create a new file
I\$Delete	Delete file
I\$Detach	Detach I/O device
I\$Dup	Duplicate path
I\$GetStt	Get path status
I\$MakDir	Make directory file
I\$Open	Open path to existing file or device
I\$Read	Read data
I\$ReadLn	Read line of text with editing
I\$Seek	Change current file position
I\$SetStt	Set path status
I\$Write	Write data
I\$WriteLn	Write line of text with editing

### 1.1.3 CD-RTOS System State Service Request Summary

All of the system state service requests available in CD-RTOS are outlined in Figure VII.3 below.

Figure VII.3 **CD-RTOS System State Service Requests**

Name	Description of Use
F\$Alarm	Send alarm signal after specified time period
F\$AIPD	Allocate a process or path descriptor
F\$AIPrc	Allocate a process descriptor
F\$AProc	Place a process in the active process queue
F\$DelPrc	De-allocate process descriptor service request
F\$FindPD	Find the address of a process or path descriptor
F\$I/OQu	Suspend the calling process in an I/O queue
F\$IRQ	Install or remove a device from the IRQ table
F\$Move	Copy a block of data
F\$NProc	Activate the next process in the active queue
F\$RetPD	Deallocate a process or path descriptor
F\$SSvc	Install system service request(s)
F\$VModul	Validate the CRC of a module in memory

### 1.1.4 Signal Handling Facilities

In CD-RTOS, a process may install a subroutine to be executed when the process receives a signal. This subroutine is called the "signal intercept routine". This subroutine is called automatically when the process receives the signal. If the process is sleeping, it is awakened.

If a process has no signal intercept routine when it receives a signal, it will be aborted. The signal intercept routine is installed by a process using the F\$lcpt service request.

If a process receives a signal while it is processing another signal, the new signal will be placed in a First In - First Out (FIFO) queue for the process. As soon as the signal intercept routine executes the F\$RTE service request, the second signal is processed.

A process may also select to mask signals for a period of time. Any signals received while signals are being masked are put into the FIFO queue. As soon as signals are unmasked, any queued signals will be processed.

VII.1 Kernel

---

The S\$Wake signal is handled as a special case. It activates a sleeping process but does not cause the intercept routine to be executed and does not abort a process which has no intercept routine. S\$Wake is never queued since the presence of any signal insures that the process is awake.

---

**VII.1 Kernel**

---

**1.2 CD-RTOS Startup Procedures**

The general outline of steps performed during CD-RTOS initialization (e.g. at power on or reset) are made up of three distinct parts. The first part consists of manufacturer dependent routines executed prior to calling the CD-RTOS kernel. The second part is made up of routines performed by the CD-RTOS kernel itself. The final part consists of operations performed by the manufacturer dependent initial process routines started by CD-RTOS.

The following is a description of the start-up procedures for CD-I players.

**Manufacturer dependent (bootstrap) routines**

1. Reset or initialize hardware
2. Perform any necessary hardware diagnostics
3. Locate (and initialize if necessary) system RAM, ROM and NVRAM
4. Locate and execute the CD-RTOS kernel

**CD-RTOS routines**

1. Build list of free RAM
2. Locate modules in ROM and build module directory
3. Initialize internal system global variables and execution vectors
4. Link to Init module
5. Start system clock
6. Execute kernel customization module, if present in ROM<sup>1</sup>
7. Open system I/O paths
8. Execute the initial process

---

<sup>1</sup> It is mandatory for a manufacturer to supply a kernel customization (os9p2) module to carry out two tasks. First, it must locate Configuration Status Descriptor modules in ROM and copy them into a file in NVRAM (if the file doesn't already exist). Secondly, it must install a service routine for F\$Panic. This routine will be called by the kernel at point 9.

VII.1 Kernel

---

9. Wait for all processes to die and then execute the F\$Panic system call installed by the os9p2 module. This routine should merely execute F\$SysDbg on base case systems. This will cause the system to jump to point 1 of the Manufacturer dependent (bootstrap) routines<sup>2</sup>

**Example of manufacturer dependent initial process routines**

1. Optionally display manufacturer dependent image
2. Try to link to the event "CSD\_ERROR". If this event exists it indicates that the csdinit module had an error in creating the CSD file. The initial process must take steps to create more space in the NVRAM and then exit. It is recommended that the initial process prompts the user to delete one or more files on the NVRAM to create additional space.
3. Wait for user to insert and play disc
4. If CD-DA disc play it and go back to 1
5. Change default working and execution directories to the CD-I disc
6. If possible read Disc Label to determine the initial application program name
7. If no Disc Label or no initial application program present on disc, display suitable message and go to 1
8. Close all open paths to UCM and CDFM
9. Chain to initial application (which must be loaded into bank B of memory).

---

<sup>2</sup> If the user plays a CD-I Disc the initial process will set up for and then chain to the application on the disc. This means that when the application exits there will be no processes running on the system. By resetting the system, this assures that any modules loaded by the application are removed and any memory used on behalf of the application is returned to the system. This guarantees that the next disc played will have full access to the CD-I system's resources.

VII.1 Kernel

---

Some CD-RTOS system modules in ROM may be replaced by an application by providing a replacement module<sup>3</sup> which has a revision number greater than the revision number of the module that is being replaced. All such replacement modules must be upward compatible with the modules that they are replacing.

---

<sup>3</sup> Although technically possible, manufacturer dependent modules (e.g. device drivers) must not be replaced using this technique. This would violate the requirement that every CD-I disc must be playable on every CD-I player. This includes the kernel.

### 1.3 Hardware Configuration Status Descriptor

#### 1.3.1 General

In order for an application program to be able to determine the functional configuration of a CD-I system, CD-RTOS provides the Configuration Status Descriptor.

The configuration status descriptor (CSD) describes the functional hardware configuration of a system. It contains one entry for each device on the system whether it is a Base Case device or an optional extension.

The CSD contains sufficient information for an application to identify the characteristics of every device in a CD-I system. The physical capabilities of a specific device must be determined by an examination of the device descriptor, or by issuing appropriate I\$GetStat requests.

Each device is represented in the CSD by its Device Status Descriptor (DSD). Each DSD is made up of 3 parts, a name, a type, and a list of parameters which describe the functional capabilities of the device.

#### 1.3.2 Compilation of the CSD

When the kernel starts up, it executes a special os9p2 module to compile the CSD. Initially, this csdinit module checks to see if there is a file on the NVRAM device called "csd". If not, it creates one. Next it searches through the module directory for data modules with a type byte set to CSDData. These data modules contain the actual DSD entries which go to make up the CSD.

Next, csdinit searches these data modules for DSD's which are not already in the CSD file, adding new ones as it finds them. At the same time it looks for DSD's which are in the file but are no longer referenced by DSD's in the ROM's. These it removes from the file. Finally, csdinit looks for the Montype constant in the system memory to find out how to set the parameters for the video device.

If csdinit is unable to extend the CSD or gets an error in accessing the CSD file or data modules, it will create a CD-RTOS event named "CSD\_ERROR". The initial value of the event will be the error number received.

---

**VII.1 Kernel**

---

**1.4 Application Startup Environment**

The exact details of the application startup environment will likely vary from one manufacturer's player to another, but there are certain minimum resource levels which may be relied upon for a base case player application.

System RAM is divided into two banks of 512K bytes each. A certain amount of this memory may be used by the video hardware. The kernel will also allocate a large block of this memory at the base of bank A. Typically, this will absorb the lowest 16K of RAM. The video driver may also allocate a small block at the highest memory address of bank A.

To balance this usage, memory for device static storage will be allocated from high memory in bank B. If more than approximately 14-16K of memory is used for device static storage and internal buffers after the base case devices have been initialized, additional requests will be balanced between the banks. Typically, the total usage of the system at startup will be less than 32K, but it is guaranteed that the total will be less than 64K.

When the player shell (which resides in ROM) begins to run, it will allocate some memory for initialized and uninitialized data. This is also likely to come from bank B.

When the player shell loads the initial application from the CD-I disc, it must first deallocate all memory from bank B. It then loads the application into bank B. This causes the application to load into the highest free address of bank B. The application will have a link count of 2, although any modules loaded with the application will have a link count of one. Bank A will have approximately 480 K free (minimum) and bank B will have about 480K free minus the size of the application (and any modules loaded with it) and the size of the application data area. Most of this memory will be in two large blocks, one from each bank.

For a more detailed description of static and dynamic memory usage by CD-RTOS see Chapter VIII.7.5.

Finally, when the initial application begins execution, it must set all parameters of its environment. Any condition having to do with the state of the graphic cursor, video display or sound control will have to be set explicitly, including the pointer or graphic cursor origin.



## VII.2 File Managers

---

### VII.2 File Managers

#### 2.1 General

A File Manager is a system memory module which handles I/O requests for a class of similar devices. File managers operate at a high level, in a device independent way. Actual control of device hardware is performed by a device driver at the request of the file manager. The function of a file manager is to remove as much device dependence as possible from the I/O stream, and present a uniform I/O interface to higher levels of software.

The concept of a file is provided by a file manager. A file is made up of a contiguous stream of bytes, with no assumptions about the organization of these bytes being made by the file manager. When possible, the bytes are made randomly accessible. On devices that support random access, files are organized into directories.

All file managers are organized as a collection of subroutines called by the operating system kernel to perform specific functions. The interface is identical for every file manager, regardless of differences in device capabilities. If a function cannot be accommodated by any particular class of devices, the file manager simply returns an error.

For a detailed specification of the file manager interface, refer to the A VII.1.

This section specifies the Compact Disc File Manager (CDFM), User Communications Manager (UCM) and Non-volatile RAM File Manager (NRF). For details of the Pipe File Manager see A VII.1.

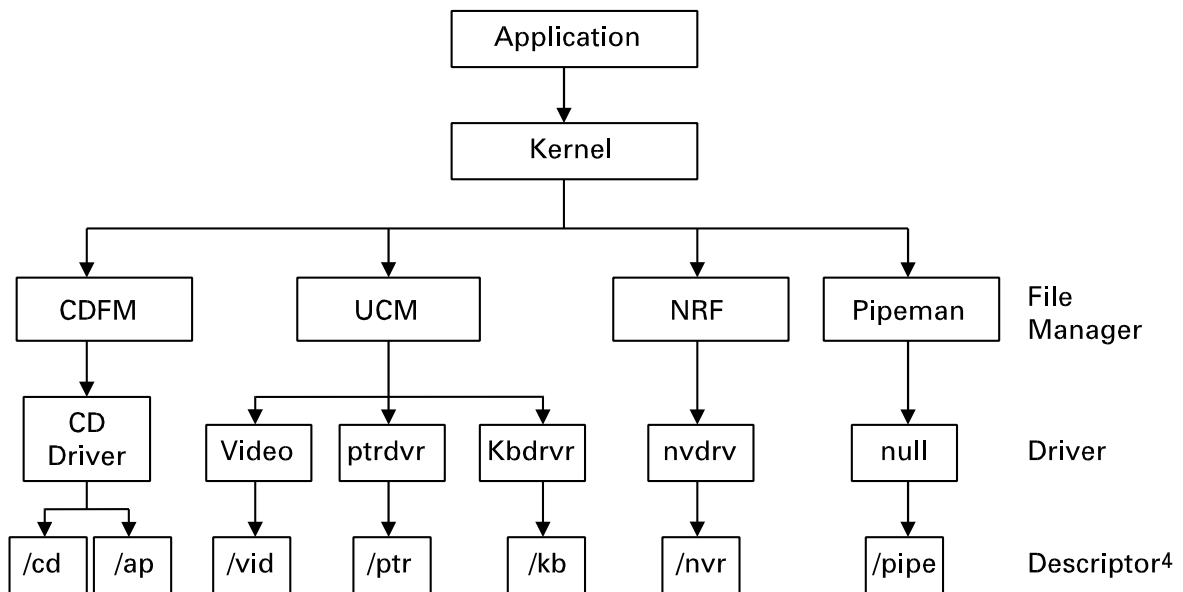
A summary of the functions provided is given in Figure VII.4 below along with a data flow model for CD-RTOS.

VII.2 File Managers

Figure VII.4a **Functions Provided by the File Managers**

Name	Description
Create	Create a new file
Open	Open an existing file
Makdir	Create a new directory file
Chgdir	Change current working directory
Delete	Delete a file
Close	Close an open file
Seek	Reposition the current file pointer
Read	Read bytes or records
Readln	Read until end of line character
Write	Write bytes or records
Writeln	Write until end of line
Getstat	Return device status (wildcard)
Setstat	Change device status (wildcard)

Figure VII.4b **Data Flow Model for CD-RTOS and its Relation to the Hardware**



<sup>4</sup> /nvr is the only mandatory name. Other device names are only examples: their real names can be obtained by consulting the **CSD**.

This page is intentionally left blank

VII.2 File Managers

---

**2.2 Compact Disc File Manager (CDFM)****2.2.1 Introduction**

The CDFM is a file manager module that supports I/O to the CD device driver. It is written as a standard file manager as described in A VII.1. The function of the CDFM is to convert high level user commands to low level CD device driver commands. The CDFM is responsible for management and access of the disc's tree structured file system, the raw data stream from the disc, and scheduling of the disc's usage.

Since the Audio Processor is an integral part of the CD interface hardware, CDFM also supports audio output and control.

The CDFM is a collection of subroutines. The CDFM supports the following subroutine entry points referred to as service requests.

**Figure VII.5 Subroutine Entry Points for the CD-File Manager**

Name	Description
Open	Open path to specified device and/or file
Chgdir	Change user's default directories
Seek	Change current file position pointer
Read	Read requested number of bytes from current file position
ReadLn	Read requested number of bytes from current file position until carriage return <CR> or maximum specified
GetStat	Get specified status information
SetStat	Set specified status information
Close	Close path to a specified device and/or file

## VII.2 File Managers

---

### 2.2.2 Features and Concepts

#### 2.2.2.1 Execution Classes

There are two execution classes for the service requests to the CDFM. The first class consists of the asynchronous execution service requests. These are service requests which, when called from an application, start the execution of the specified I/O function and return to the application while the service request is executing. This allows the application and service request to execute concurrently. These service requests include four of the SetStat service requests (defined fully in VII.2.2.3.2): SS\_Play, SS\_CDDA, SS\_Seek and SM\_Out (VII.2.2.4.1).

The second class consists of the synchronous execution service requests. These are service requests which, when called from the application, complete their function before returning to the application. The synchronous service requests include all other functions not included in the asynchronous service requests.

#### 2.2.2.2 Scheduling

The CDFM schedules the use of the CD-I Disc Drive. The scheduling algorithm used is the 'elevator' or 'SCAN' seek optimization strategy. This is a strategy that minimizes the total seek times by moving the disc head back and forth across the disc surface, servicing all requests in its path. The head changes direction only when there are no more requests to service in the current direction.

**Note:** If a disc access request is currently executing, any requests to access the disc from other processes will be queued until the disc becomes available. If the second request is an asynchronous request, it will not return to the application until that request actually begins execution.

If the same process executes a second disc access request while its first disc access request is still active, it will receive an error indicating that the device is busy. The same process may not have two disc access requests pending at the same time on the same path.

## VII.2 File Managers

---

### 2.2.2.3 File Security System

File usage and security are based on file attributes. Each file can have up to seven possible attributes (directories can have eight).

The term permission is used when one of the eight possible attribute characters is set. Permission determines who can access a file or directory and how it can be used.

Permissions are based on a group and user number associated with each file and with the process trying to access the file. The group number 0 is a special group called the "super user group". A super user has access to any file in the system.

For a description of the different permissions bits and their meaning, refer to III.21.

In a CD-I system, each application will initially be executed as user 0.0. That is, group 0, user 0. If that application is to access a file, the file must have the permission bits set which would allow that group/user to access it.

This gives the primary application the ability to access any file on the disc. If an application wishes to restrict itself or to fork a restricted application, it may change its own group or user number, thus limiting itself to only those files with appropriate permissions for that group and user number.

### 2.2.2.4 Audio Scheduling

There are three functions which send audio to the audio processor. Two of them play ADPCM audio (SM\_Out and SS\_Play) and one plays PCM audio (SS\_CDDA). Since they are all asynchronous they may interact in different ways. SS\_Play and SS\_CDDA both require disc access and so are covered by the discussion of disc access scheduling in VII 2.2.2.2.

When audio is being output from memory concurrently with ADPCM audio from the disc, the audio from memory has precedence. The audio from the Real Time File will not be heard until the soundmap is done, regardless of which was initiated first.

If SM\_Out is executed while CD-DA audio is playing, the CD-DA play will be aborted.

If SS\_CDDA is executed while a soundmap is playing, the soundmap will be aborted after the current sector (18 sound groups) has finished playing.

## VII.2 File Managers

---

### 2.2.2.5 Directory Sector Buffering

To improve the performance of the I\$Open operation, directory sectors are buffered by CDFM much as normal Reads are buffered for standard files. A one sector buffer is used to contain the last directory sector read by CDFM.

All directory searching starts by reading the first directory sector, so if a directory is contained entirely within one sector, multiple I\$Open calls to files within that directory will not cause multiple disc access operations.

On average, one directory sector will contain about 30 file descriptors. So for the greatest performance, a disc preparer should keep the number of files in a directory small enough that only one sector is used to contain the file descriptors for that directory, and the application should open all necessary files within a directory before opening files in other directories.

## VII.2 File Managers

---

### 2.2.3 CD Service Requests

This is a specification of the interface between the application program and the CDFM for the CD device.

#### 2.2.3.1 **ISGetStt - GetStat Functions**

These functions are used to get specific status information.

The service requests performed by the GetStat entry point in the CDFM are specified by a function code and are described in Figure VII.6.

Figure VII.6 **GetStat Service Requests in the CDFM**

Name	Description
SS_Opt	Get options section of path descriptor
SS_EOF	Test for end of file
SS_Size	Return file size
SS_Pos	Return file position
SS_CDFD	Return file descriptor
SS_Path	Return full pathlist

**Note:** Any function codes other than the above are passed through CDFM to the CD device driver GetStat routine for possible action there. This allows direct access to functions supported by the device driver (see VII.3.1.5.4 for more about GetStat in the device driver).



**SS\_Opt - Get Options Section of Path Descriptor**

This call returns a 128-byte packet which contains the options section of the path descriptor. The options section of the path descriptor contains information (see VII.3.1.4.2.3 and Appendix VII.1) which may be of use to the application. Some of the fields in the options section may be changed by the application. To do this, the options section is normally read (using this service request), the changes are made, and the options section is rewritten using the SS\_Opt SetStat call. Changing PD\_CNUM selects a different channel mask for the next synchronous CD disc I/O operation.

Input:           d0.w = Path number  
                  d1.w = SS\_Opt getstat code  
                  (a0) = Address of 128 byte buffer

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**SS\_EOF - Test for End of File**

This call is used to determine if the current file pointer is pointing at the end of the file. If it is at the end error code E\$EOF is returned.

Input:           d0.w = Path number  
                  d1.w = SS\_EOF getstat code

Output:          d1.l = 0 if not end of file

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$EOF

**SS\_Size - Return File Size**

This call is used to get the total size, in bytes, of the file denoted by the path number. The file size returned is read from the file size in the path descriptor. This is originally determined by taking the number of sectors occupied by the file multiplied by 2048 less any unused portion of the last sector of the file. The file size does not take into account the fact that form 2 sectors actually contain 2324 bytes of data and CD-DA tracks are also of a different size.

Input:           d0.w = Path number  
                  d1.w = SS\_Size getstat code

Output:           d2.l = File size

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**SS\_Pos - Return File Position**

This function returns the position of the next byte to be read from the file.

Input:           d0.w = Path number  
                  d1.w = SS\_Pos getstat code

Output:          d2.l = Current file position

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**SS\_CDFD - Return File Descriptor**

This call is used to read the file descriptor describing the path number. The file descriptor may be read for information purposes only as there are no user changeable parameters (see III.3.2 for a complete specification of the file descriptor).

Input:           d0.w = Path number  
                  d1.w = SS\_CDFD getstat code  
                  d2.w = Number of bytes to copy  
                  (a0) = Pointer to buffer area for file descriptor

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**SS\_Path - Get Full Pathlist to Open File**

This call returns the complete pathlist to the open file, including the device name, in a 128-byte buffer.

Input:           d0.w = Path number  
                  d1.w = SS\_Path getstat code  
                  (a0) = Pointer to 128-byte buffer

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$BTyp, E\$DevBsy, E\$BPNam, F\$SRqMem

VII.2 File Managers

---

**2.2.3.2 I\$SetStt - SetStat Functions**

Set specific status information.

The service requests performed by the SetStat entry point in the CDFM are specified by a function code and are outlined in Figure VII.7.

Figure VII.7 **SetStat Service Requests in the CDFM**

Name	Description
SS_Seek	Issue a seek command to disc drive
SS_Abort	Abort asynchronous operation in progress
SS_CDDA	Play CD digital audio
SS_Pause	Pause the disc drive
SS_Eject	Issue door open command to disc drive
SS_Mount	Mount disc by disc number
SS_Opt	Write the options section of the path descriptor
SS_Disable	Disable front panel buttons
SS_Enable	Enable front panel buttons
SS_Cont	Continue the disc after pause
SS_Stop	Stop the disc
SS_Raw	Read Raw sectors
SS_ReadTOC	Return TOC from Q channel
SS_Play	Play real time records
SS_CChan	Channel and Audio channel changing

**Note:** Any function codes other than the above codes are passed through CDFM to the CD device driver SetStat routine for possible action there. This allows direct access to functions supported by the device driver (see VII.3.1.5.4. of the CD-I Specification for more about SetStat in the device driver).

**SS\_Seek - Issue a Seek Command to Disc Drive**

This function causes the file position pointer to be set to the new value. It also causes the disc drive to physically seek to the beginning of the sector containing the specified byte. The SS\_Seek command uses a sector size of 2048 bytes to compute the new file position. This function is an asynchronous operation and therefore uses a status block to keep the application informed of the status of the seek.

The format of the status block is given in Figure VII.8.

Figure VII.8 **Structure of a Status Block**

Offset	Length	Name	Description
0	2	ASY_Stat	Current status of the operation
2	2	ASY_Sig	Signal to be sent on termination

**ASY\_Stat:** This field contains the current status of the asynchronous operation. It should be set to zero by the application. The status defines whether the operation is finished and whether there was an error during the operation. The system updates this field and it should not be altered by the application. The bit assignments for this field are as follows.

Figure VII.9 **ASY\_Stat Field**

Bit #	Description (if set)
0	Operation is finished
1-14	Reserved (must be zero)
15	Error has occurred



VII.2 File Managers

---

ASY_Sig:	This field contains the signal number to be sent to the application when the operation is finished or an error occurs. It is initialized by the application before the operation is started and may be changed by the application during the operation. If this field is set to zero then no signal is sent when the operation is finished or when an error occurs. If an error occurs during the operation this field contains the error code of the error.
Input:	d0.w = Path number d1.w = SS_Seek setstat code d2.l = New byte position within file (a0) = Pointer to status block
Output:	None
Error Output:	cc = Carry bit set to one d1.w = Error code
Possible Errors:	E\$BPNum, E\$DevBsy

**SS\_Abort - Abort Operation in Progress**

This function is used to abort a CD operation currently in progress. This is a privileged function call and may be used only by a process with a user ID of the super user or with the user ID of the process which started the operation. If a synchronous request is in progress an E\$Abort error is returned. If an asynchronous request is in progress any signal in PCB\_Sig (or ASY\_Sig) will be sent and an E\$Abort error code is returned in PCB\_Sig (or ASY\_Sig). After the signal is sent, both the Done bit and the Error bit will be set in PCB\_Stat (or ASY\_Stat). The SS\_Abort function will return with carry clear.

Input:           d0.w = Path number  
                  d1.w = SS\_Abort setstat code

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$Permit

**SS\_CDDA - Play CD Digital Audio**

This function is used to play CD-DA sectors on the disc. The play starts from the current file position and continues until an error occurs, the maximum number of sectors has been played or the end of the file has been reached. If a soundmap is being output, then that output is discontinued after the current sector (18 sound groups) has finished playing. This function is an asynchronous operation and therefore uses a status block to keep the application informed of the status of the SS\_CDDA. The format of the status is identical to the status block of SS\_Seek. It should be noted that when playing CD-DA the resolution of the current file position can only be guaranteed to  $\pm 75$  sectors, i.e.  $\pm$  one second. The file position pointer is updated whenever a selected sector is retrieved from disc.

Input:           d0.w = Path number  
                  d1.w = SS\_CDDA setstat code  
                  d2.l = Maximum number of sectors to play  
                  (a0) = Address of status block

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$DevBsy

**SS\_Pause - Pause the Disc Drive**

This function is used to pause the CD disc drive. This is a privileged function call and may be used only by a process with a user ID = 0.0 (see A VII.1) or with the user ID of the process which started an asynchronous operation currently in progress. On CD-I tracks and on CD-DA tracks, the disc will pause at the position on the disc being played when the pause was requested.

Input:           d0.w = Path number  
                  d1.w = SS\_Pause setstat code

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$Permit

**SS\_Eject - Issue Door Open Command to Disc Drive**

This function is used to open the CD disc drive door. If an operation is in progress, the effect of an SS\_Abort is done before the eject command is issued. This is a privileged function call and may be used only by a process with a user ID = 0.0.

Input:           d0.w = Path number  
                  d1.w = SS\_Eject setstat code

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$Permit, E\$UnkSvc

**SS\_Mount - Mount Disc by Disc Number**

This call changes the default disc to use in multidisc players. The disc is referred to by a number with maximum range of 0 to 65535. On single disc players the disc number used is 0. SS\_Mount also causes the drive door to close if possible and spins up the disc.

Input:           d0.w = Path number  
                  d1.w = SS\_Mount setstat code  
                  d2.w = Disc number

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**SS\_Opt - Set Options Section of Path Descriptor**

This call writes a 128-byte packet which contains the options section of the path descriptor. The options section of the path descriptor contains information (see VII 3.1.4.2.3 and A VII.1) which may be of use to the application. The options section is normally read using the GetStat function call SS\_Opt. The changes are made and the options section is rewritten using this service request. Currently, only PD\_CNUM may be changed. Changing PD\_CNUM selects a different channel mask for the next I\$Read or I\$ReadLn on a CD-I disc.

Input:           d0.w = Path number  
                  d1.w = SS\_Opt setstat code  
                  (a0) = Pointer to 128-byte packet

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNUM

**SS\_Disable - Disable Hardware Control Buttons on Player**

This call disables any disc drive related user controls.

Input:           d0.w = Path number  
                  d1.w = SS\_Disable setstat code

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum



**SS\_Enable - Enable Hardware Control Buttons on Player**

This call enables any disc drive related user controls.

Input:           d0.w = Path number  
                  d1.w = SS\_Enable Setstat

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**SS\_Cont - Continue the Disc Drive**

This function continues the playing by the disc drive if the drive was paused.

Input:           d0.w = Path number  
                  d1.w = SS\_Cont setstat code

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**SS\_Stop - Stop the Disc Drive**

This function is used to stop the disc drive. This function works the same as the SS\_Eject function, except that the drive door is not opened.

Input:           d0.w = Path number  
                  d1.w = SS\_Stop setstat code

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$Permit

**SS\_Raw - Read Raw Sectors**

SS\_Raw returns the sector contents from byte number 12 to byte number 2351 of the raw sector data. This includes the header, subheader, data, and ECC/EDC, if any. The SS\_Raw function works identically to the Read function with the exception that both Mode 1 and Mode 2, Form 1 and Form 2 sectors may be read and that the size of the data returned per sector is 2340. The current file position must be on a sector boundary when this call is made. The file position pointer is incremented by 2048 for every sector read.

Input:                   d0.w = Path number  
                          d1.w = SS\_Raw setstat code  
                          d2.l = Maximum number of bytes to read (must be in  
                                  increments of 2340)  
                          (a0) = Buffer pointer

Output:                   d1.l = Number of bytes actually read

Error Output:           cc    = Carry bit set to one  
                          d1.w = Error code

Possible Errors: E\$BPNuM, E\$Param, E\$Abort, E\$NotRdy, E\$Read, E\$EOF,  
E\$DevBsy

**SS\_Read\_TOC - Return TOC from Q-Channel**

SS\_Read\_TOC returns the Table Of Contents (TOC) from the Q-channel in the user's buffer.

Input:           d0.w = Path number  
                  d1.w = SS\_Read\_TOC setstat code  
                  d2.l = Number of bytes to return (maximum 510)  
                  (a0) = Pointer to buffer for TOC

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$Read, E\$DevBsy, E\$Abort, E\$NotRdy

## VII.2 File Managers

Figure VII.10 Table Of Contents (TOC) Structure Format

Offset	Length	Name	Description
0	1	TOC_CTRL0	Control <sup>5</sup> byte 0
1	1	TOC_A0	Contains \$A0
2	1	TOC_STN0	PMIN of Point \$A0 <sup>6</sup>
3	1	TOC_TYPE	PSEC of Point \$A0 <sup>6</sup>
4	1	TOC_PBLK0	PFRAME of Point \$A0 <sup>6</sup>
5	1	TOC_CTRL1	Control <sup>5</sup> byte 1
6	1	TOC_A1	Contains \$A1
7	1	TOC_LTN0	PMIN of Point \$A1 <sup>6</sup>
8	1	TOC_PSEC1	PSEC of Point \$A1 <sup>6</sup>
9	1	TOC_PBLK1	PFRAME of Point \$A1 <sup>6</sup>
10	1	TOC_CTRL2	Control <sup>5</sup> byte 2
11	1	TOC_A2	Contains \$A2
12	1	TOC_LMIN	Start minute of LEAD OUT area
13	1	TOC_LSEC	Start second of LEAD OUT area
14	1	TOC_LBLK	Start block of LEAD OUT area
15	1	TP_CTRL	Control <sup>5</sup> byte of first track entry in TOC list <sup>7</sup>
16	1	TP_TNO	Track number in 2 digits BCD (8-4-2-1 code): \$01 for track 01
17	1	TP_\$MIN	Track 1 start minute (2 digits BCD)
18	1	TP_\$SEC	Track 1 start second (2 digits BCD)
19	1	TP_\$BLK	Track 1 start block (2 digits BCD)
Contains 97 identical packets (like offset 15-19), one for each successive track entry in TOC list <sup>7</sup>			
505		TP_CTRL	Control <sup>5</sup> byte
506		TP_TNO	Track number
507		TP_\$MIN	Start minute
508		TP_\$SEC	Start second
509		TP_\$BLK	Start block

<sup>5</sup> CTRL + ADDR fields as defined in CD-DA Specifications

<sup>6</sup> For detailed contents see II.2.5 and Figure VII.11

<sup>7</sup> Only packets corresponding to track entries in the TOC list are valid. Other packets are undefined. The number of valid packets = \$A1.PMIN - \$A0.PMIN + 1

Figure VII.11 **Table of Disc Types**

DISC TYPE	\$A0.PMIN	\$A0.PSEC
CDI	$\geq 2$	= \$10
CD-ROM	$\geq 1$	= \$00
CD-ROM XA	= 1 = 1 $\geq 1$	= \$00 or = \$20
CD-DA		= \$00

**SS\_Play - Play Real-time Records**

SS\_Play is used to play data off the disc. It can be used to 'play' Mode 2 Form 1 and Mode 2 Form 2 sectors on the disc. The play function is normally used to play real time records. A play is controlled by information contained in the play control block (PCB). The play control block is built by the application. It contains information such as number of records to play, which channel numbers to select, which data types to select, what signal to send on Real-time events, etc. Play is an asynchronous operation (i.e. the application continues execution at the same time as the play is executing). The layout of the play control structures define the functionality of play. The play is started using data pointed to by the file position pointer. The file position pointer is updated whenever a selected sector is retrieved from disc.

Input:           d0.w = Path number  
                  d1.w = SS\_Play setstat code  
                  (a0) = Pointer to play control block (PCB)

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$DevBsy

The address of the play control block is passed as a parameter to the SS\_Play function. The play control block contains the status of the play, the signal to send when the Play terminates, total number of records to play, channel numbers to select and pointers to lists of Channel Index List (CIL) for each data type.



## VII.2 File Managers

---

The values within the play control block, except PCB\_Stat, can be changed at any time by the application. They will take effect after the next selected sector (according to previous PCB\_Chan value) is retrieved from disc<sup>8</sup> or after the reception of an EOR or Trigger bit.

If PCB\_Rec was set to 0 by the application, the play is aborted after processing that sector.

Consequently, the effectiveness of channel selection switching by using PCB\_Chan may be more or less immediate depending on the file organization: this uncertainty is most of the time acceptable, especially for the switch between various audio channels.

However, when immediate and precise channel switching is required (for instance synchronized on a trigger bit, between exclusive video or data channels) the channel selection switch can be made through one of two mechanisms. First, the SS\_CChan setstat will cause the PCB\_Chan and PCB\_AChan masks to be updated. Secondly, the CIL can be used to ignore data from specific channels. Placing a zero (instead of the PCL pointer) into the CIL entry will cause the data for that channel to be discarded (see Figure VII.14).

---

<sup>8</sup> A sector will be selected from disc if its subheader field fullfills the 2 following conditions:

1. The 'file number' field in the sector subheader matches the 'file number' of the current file descriptor: in other words, the sector belongs to the file being presently accessed.
2. The sector subheader contains a 'channel number' field which matches the effective PCB\_Chan value.

**Play Control Block (PCB) Format**Figure VII.12 **Play Control Block Format**

Offset	Length	Name	Description
0	2	PCB_Stat	Current status of the Play
2	2	PCB_Sig	Signal to be sent on termination
4	4	PCB_Rec	Maximum number of records to play
8	4	PCB_Chan	Channel selection mask
12	2	PCB_AChan	Audio to audio processor channel selection mask
14	4	PCB_Video	CIL pointer for video data
18	4	PCB_Audio	CIL pointer for audio data
22	4	PCB_Data	CIL pointer for computer data

The PCB parameters are defined as follows.

**PCB\_Stat:** This field contains the current status of the Play. It should be set to zero by the application. This field is updated by the system at the time a sector is selected. The application may review this field at any time during the Play to determine the current status of the Play but should not modify the field during the Play. The bits in PCB\_Stat are defined as follows:

Bit Number	Description (if set to one)
0	End of Record (EOR)
1	Video sector last read
2	Audio sector last read
3	Data sector last read
4	Trigger (T)
5	Form 2
6	Real-Time Sector (RT)
7	End-of-File (EOF)
8	Play is finished
9-14	Reserved (must be zero)
15	Error has occurred

VII.2 File Managers

---

- PCB\_Sig:** This field contains the signal number to be sent to the application when the Play is finished, or on EOR or trigger bit. It is initialized by the application before the Play is started and may be changed by the application during the Play. If this field is set to zero then no signal is sent when the Play is finished or when an error occurs. If an error occurs during the Play this field contains the error code of the error.
- PCB\_Rec:** This field contains the maximum number of real time records left to play. The application initializes this field to the maximum number of real time records to Play. The system decrements this number every time a record is Played until the field goes to zero at which time the Play is terminated. This number may be changed by the application during the execution of the Play.
- If PCB\_Rec is set to 0, the play will be aborted after the next selected sector is processed.
- Receipt of an End Of File bit, in and of itself, will not affect the value of this field.
- PCB\_Chan:** This field contains the channel selection mask. It defines which channels are played. Any combination of 32 separate channels may be played by setting the corresponding bit in this field. This field may be changed by the application during the execution of the Play.
- PCB\_AChan:** This is a channel selection mask to select audio sectors which go to the audio processor. If a bit in this field is set to one, the audio sectors with the corresponding channel number will be sent to the audio processor. All channels included in this field must also be defined in PCB\_Chan. Channel bits set to one which are included in PCB\_Chan but not in this field will be handled by the PCL for audio sectors if one exists.
- PCB\_Video:** Points to the Channel Index List (CIL) for video data. If this field contains zero, video data blocks are ignored. This field is initialized before the Play is started by the application.

VII.2 File Managers

---

**PCB\_Audio:** This points to the Channel Index List (CIL) for audio data. If this field contains zero, audio data blocks are ignored. This field is initialized before the Play is started by the application.

**PCB\_Data:** This points to the Channel Index List for program related data. If this field contains zero, program related blocks are ignored. This field is initialized before the Play is started by the application.

**Channel Index List (CIL) Format**

There are three Channel Index List pointers in the PCB, one each for audio, video, and program related data blocks. The Channel Index List contains pointers to lists of Play Control Lists (PCLs), according to channel numbers. CIL has two different formats, i.e. one is for audio and another is for video or program related data. This difference is based on the difference of available channel numbers for each type of data.

Figure VII.13 **CIL for Audio data**

Offset	Length	Name	Description
0	4	CIL_00	PCL pointer for channel = 0
4	4	CIL_01	PCL pointer for channel = 1
8	4	CIL_02	PCL pointer for channel = 2
12	4	CIL_03	PCL pointer for channel = 3
16	4	CIL_04	PCL pointer for channel = 4
20	4	CIL_05	PCL pointer for channel = 5
24	4	CIL_06	PCL pointer for channel = 6
28	4	CIL_07	PCL pointer for channel = 7
32	4	CIL_08	PCL pointer for channel = 8
36	4	CIL_09	PCL pointer for channel = 9
40	4	CIL_10	PCL pointer for channel = 10
44	4	CIL_11	PCL pointer for channel = 11
48	4	CIL_12	PCL pointer for channel = 12
52	4	CIL_13	PCL pointer for channel = 13
56	4	CIL_14	PCL pointer for channel = 14
60	4	CIL_15	PCL pointer for channel = 15

Figure VII.14 **CIL for Video/Program Related Data**

Offset	Length	Name	Description
0	4	CIL_00	PCL pointer for channel = 0
4	4	CIL_01	PCL pointer for channel = 1
8	4	CIL_02	PCL pointer for channel = 2
12	4	CIL_03	PCL pointer for channel = 3
16	4	CIL_04	PCL pointer for channel = 4
20	4	CIL_05	PCL pointer for channel = 5
24	4	CIL_06	PCL pointer for channel = 6
28	4	CIL_07	PCL pointer for channel = 7
32	4	CIL_08	PCL pointer for channel = 8
36	4	CIL_09	PCL pointer for channel = 9
40	4	CIL_10	PCL pointer for channel = 10
44	4	CIL_11	PCL pointer for channel = 11
48	4	CIL_12	PCL pointer for channel = 12
52	4	CIL_13	PCL pointer for channel = 13
56	4	CIL_14	PCL pointer for channel = 14
60	4	CIL_15	PCL pointer for channel = 15
64	4	CIL_16	PCL pointer for channel = 16
68	4	CIL_17	PCL pointer for channel = 17
72	4	CIL_18	PCL pointer for channel = 18
76	4	CIL_19	PCL pointer for channel = 19
80	4	CIL_20	PCL pointer for channel = 20
84	4	CIL_21	PCL pointer for channel = 21
88	4	CIL_22	PCL pointer for channel = 22
92	4	CIL_23	PCL pointer for channel = 23
96	4	CIL_24	PCL pointer for channel = 24
100	4	CIL_25	PCL pointer for channel = 25
104	4	CIL_26	PCL pointer for channel = 26
108	4	CIL_27	PCL pointer for channel = 27
112	4	CIL_28	PCL pointer for channel = 28
116	4	CIL_29	PCL pointer for channel = 29
120	4	CIL_30	PCL pointer for channel = 30
124	4	CIL_31	PCL pointer for channel = 31

**CIL\_##:** This points to the next Play Control List for the data whose channel number = ##. If this field contains zero, the data blocks having that channel number are ignored. This field is initialized before the Play is started by the application and is updated by the system to point to the next PCL in the list as each PCL is filled. This field may be changed by the application.

The format of each Play Control List is defined in Figure VII.15 and each field is further specified below.

**Play Control List (PCL) Format**

There are Play Control List pointers in the CIL, according to channel number and data type. Each points to a structure list that controls the destinations of the data. This structure is a singly linked list of PCL's and may be built as a circular or sequential list.

Figure VII.15 **Structure of a Play Control List**

Offset	Length	Name	Description
0	1	PCL_Ctrl	Control byte
1	1		Reserved
2	1	PCL_SMode	Submode byte
3	1	PCL_Type	Coding Information byte
4	2	PCL_Sig	Signal to be sent on buffer full
6	4	PCL_Nxt	Pointer to next Play Control List
10	4	PCL_Buf	Pointer to buffer
14	4	PCL_BufSz	Size of buffer
18	4	PCL_Err	Pointer to error buffer
22	2		Reserved
24	4	PCL_Cnt	Current offset

**PCL\_Ctrl:** This field is initialized to zero by the application before the Play is started.

**Bit 0 Buffer full**

If bit 0 is set to one, it indicates that the buffer pointed to by this PCL is full. If the system needs to use this buffer and the buffer full bit is set to one, then the system terminates the Play and returns an overrun error. The system sets this bit to one if it determines that the buffer of this PCL is full.

**Bit 7 Data Error**

Bit 7 is set to one when an error has been detected in the data transferred in the buffer. Additional information can be provided: see description of PCL\_Err. The application must then clear this field before the buffer can be re-used.

VII.2 File Managers

---

- PCL\_SMode:** This field is updated with the value contained in the submode byte of the subheader of the last sector read into this PCL. This field must not be changed at any time by the application during play.
- PCL\_Type:** This field is set by the system and contains the coding information byte of the subheader of the last sector read into the buffer of this PCL. This field should be used by the application for reference only and should not be changed by the application.
- PCL\_Sig:** This field is initialized by the application and contains the signal number to be sent to the application when the buffer of the PCL is full or an error has been detected in the data transferred in the buffer. When the PCL and PCB signals are to be sent at the same time, the PCL signal is sent first, followed by the PCB signal. This field may be changed by the application at any time during the execution of the Play. If this field contains zero, then no signal is sent.
- PCL\_Nxt:** This is the pointer to the next PCL element. The linked list of PCLs, which is built by the application, contains at least one element. The list may be circular or may be terminated by the entry of a zero in this field. The list of PCL's may be manipulated by the application at any time during the execution of the Play.
- Note: When the buffer for a PCL is full, the contents of this field is placed in the CIL entry which references the given PCL.
- PCL\_Buf:** This is the address of the buffer in which the data from the disc is to be stored. It is the responsibility of the application to make sure that this points to an allocated data area large enough to hold all the data requested for this PCL. If this pointer is zero, the data is not transferred into memory and PCL\_Cnt is not incremented. This field is initialized by the application and may be changed by the application at any time during the execution of the Play.

## VII.2 File Managers

**PCL\_BufSz:** This specifies the size, in sectors, of the buffer<sup>9</sup> pointed to by PCL\_Buf. This field is initialized by the application and may be changed by the application at any time during the execution of the Play.

**PCL\_Err:** This field points to the error information structure (shown in the table below). If this field contains a zero, (No error information structure available) only the "Data error" bit is set in PCL\_Ctrl, otherwise information about the data in error is also returned in the given buffers. These data can be used to conceal errors in video sectors for instance (see V.4.7.3).

In both cases the buffer is filled and the Play continues: it is up to the application to "abort" the play, if necessary.

Figure VII.16 Error Information Structure

Offset	Length	Name	Description
0	4	Err_BufSz	Total size of error buffer
4	1	Err_Res	Resolution (b/w) of error info.
5	1		Reserved
6	2	Err_Cnt <sup>10</sup>	No. of block(s) containing error(s)
8	2	Err_Offset	Offset to first error data block (e.g. 26+N)
10	8		Reserved for system use <sup>10</sup>
18	8		Reserved for application use
26	N	Err_blocks <sup>10</sup>	Block error Map
26+N	M		First error data block

**Err\_BufSz:** This field is used by the application to store the total size of the error information structure. The CD driver uses the buffer size to determine the maximum number of error data blocks to return. It is possible to reserve no space for error data blocks, in which case only one error flag bit per block is returned.

<sup>9</sup> The physical data size of sector data transferred in memory depends on whether the sector type is Form 1 (2048 bytes), or Form 2 Video (2324 bytes) or Form 2 Audio (2304 bytes since 20 padding bytes are ignored).

<sup>10</sup> These fields must be initialized to zero by the application, before SS\_Play is called.



VII.2 File Managers

---

**Err\_Res:** This byte is set by the CD driver to the resolution size of one bit in the error data block(s). It is set by the driver to contain either 1 or 2, for byte or word resolution, respectively.

**Err\_Cnt:** This word is set by the CD driver to the number of blocks that contain any errors. If no errors are encountered, this field contains zero, and the contents of the block error map and error data blocks are undefined.

**Err\_Offset:** This field must be set by the application to contain the offset from the beginning of the error information structure to the first error data block. If no error data blocks are to be returned, this field may contain zero. The offset of the first error data block must be an even number.

**Err\_blocks:** This field contains a bitmap with one bit for each error data block in the error information structure. If the bit is clear, i.e. zero, the block was read without ANY ERRORS. If the bit is set to one, at least one soft error was detected in the block. The amount of memory (in bytes) reserved for the Err\_blocks table must be at least the number of blocks to be read divided by eight, rounded up to the nearest even integer. The application needs to clear the table before passing it to CDFM. If NO ERRORS occur the contents of the table are undefined.

**Error data blocks:** The error data block section is used to indicate which specific bytes or words in a sector are in error. The format of an error data block is a bitmap. Each byte or word (depending on the value of Err\_Res) in the sector containing errors is represented by one bit in the error data block. If the bit is set to one, the corresponding byte or word in the data contains an error. The bitmap contains error bits for the header, subheader, data and EDC (reserved) fields of the sector.

The error data block section consists of an array of error data blocks. When the CD driver detects that the array is unable to hold another complete block it will stop producing error data. In this case, only the block error map may be used to determine which sectors contain errors.

## VII.2 File Managers

---

The application must allocate a buffer large enough to hold 294 bytes of error data for each sector. However, it is possible that the driver will only use 147 bytes for each sector depending on whether the resolution of the CD controller is byte or word.

The application does not need to clear the error data block buffer area before passing it to CDFM. Unused error data blocks are not initialized by the CD driver and their content is undefined.

**PCL\_Cnt:** This field contains the offset in bytes into the buffer of the next position to copy data. It should be initialized to zero by the application before the start of a Play. This field is updated by the system, but it may be changed by the application at any time during the execution of the Play.

### **Play Termination:**

Play is terminated by any of the following conditions:

1. EOF is encountered.
2. PCB\_Rec goes to zero.
3. A software abort is received (SS\_Abort, SS\_Eject).
4. A hardware or software fatal error occurs (error in header, etc.)

**SS\_CChan - Channel & Audio Channel Changing**

This function causes a change of channel (PCB\_Chan) or Audio channel (PCB\_AChan) selection masks. The change will take effect after the current or after the next sector from disc, regardless of the contents of the subheader.

Generally, this function is used in conjunction with SS\_Play. However, if there is also an SM\_Out active during the SS\_Play, this call will have no effect on the PCB\_AChan mask until after the termination of the soundmap.

Input:           d0.w = Path number  
                  d1.w = SS\_CChan setstat code

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$NoPlay

VII.2 File Managers

---

**2.2.3.3 \$Seek - Change Path's Current File Position**

This function causes the file position pointer to be set to the new value. This does not cause a physical seek of the head on the player. The seek command uses a sector size of 2048 bytes to compute the new file position.

Input:           d0.w = Path number  
                  d1.l = New file pointer byte position

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$BMode

VII.2 File Managers

---

**2.2.3.4 I\$Read - Read Requested Bytes From Current File Position**

Read reads the specified number of bytes from the file specified by the path number. Read can be used to read Mode 1 and Mode 2, Form 1 sectors<sup>11</sup>. Read returns only the first 2048 bytes after:

- the subheader for Mode 2 Form 1 sectors
- the header for Mode 1 sectors

Read makes use of the file and channel selection mechanisms to select the sectors to be read from the disc. This file number is set to the file number specified by the file descriptor of the open path. The channel mask is set to \$FFFFFFFF when the file is opened. This causes the selection of sectors with any channel mask in the subheader. The channel mask may be changed by rewriting the channel mask in the options section of the path descriptor using the SetStat system call SS\_Opt. If there is not enough data in the file to satisfy the Read request, fewer bytes are returned than requested. When all data in the file has been read, the next read call on the path returns an end of file error.

**Note:** It is not allowed to use I\$Read to read Mode 2 Form 2 sectors. If an attempt is made to read a Mode 2 Form 2 sector, an E\$Read error will be returned. To read Mode 2 Form 2 sectors, use SS\_Raw or SS\_Play.

Input:                   d0.w = Path number  
                          d1.l = Maximum number of bytes to read  
                          (a0) = Pointer to input buffer

Output:                   d1.l = Number of bytes actually read

Error Output:           cc = Carry bit set to one  
                          d1.w = Error code

Possible Errors:        E\$BPNum, E\$BMode, E\$Read, E\$EOF, E\$DevBsy,  
                          E\$Abort, E\$NotRdy, F\$Sleep

---

<sup>11</sup> It is recommended that the Mode 2 Form 1 sectors to be accessed through I\$Read be declared as "non-Real Time" (RT bit = 0): this causes possible errors to be corrected by the base case system, using the ECC field described in II.4.7.3.

VII.2 File Managers

---

**2.2.3.5 I\$ReadLn - Read Requested Number of Bytes from Current File Position Until Character \$OD or Maximum Requested**

ReadLn functions the same as Read with the following exception: ReadLn reads data from the file until \$OD is encountered or the requested number of bytes is read; whichever comes first.

Input:           d0.w = Path number  
                  d1.l = Maximum number of bytes to read  
                  (a0) = Pointer to input buffer

Output:           d1.l = Number of bytes actually read

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$BMode, E\$Read, E\$EOF, E\$DevBsy, E\$Abort,  
E\$NotRdy, F\$Sleep

**2.2.3.6 I\$ChgDir - Change User's Default Directories**

ChgDir changes one or both of the two default directories to the directory specified by the path list. The two default directories are the execution directory and the working directory (see the RBF (Random Block File manager) section of Appendix VII.1). The access mode byte is used to determine which of the two directories is being changed. Bit number 0 of the access mode denotes the working directory when set to one. Bit number 3 of the access mode denotes the execution directory when set to one.

Input:           d0.b = Access mode  
                 (a0) = Pointer to path list

Output:           (a0) = Updated past pathname

Error Output:   cc    = Carry bit set to one  
                 d1.w = Error code

Possible Errors: E\$BMode, E\$BNam, E\$BPNam, E\$BTyp, E\$DevBsy, E\$FNA,  
E\$PNNF, F\$SRqMem

VII.2 File Managers

---

**2.2.4 Audio Functions**

This section specifies the CDFM Audio Getstat and Setstat functions.

**2.2.4.1 Soundmap Control Functions****SM\_Creat - Create Soundmap**

SM\_Creat creates a soundmap.

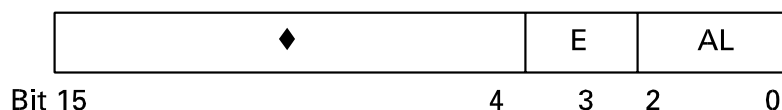
Input:           d0.w = Path number  
                   d1.w = SS\_SM setstat code  
                   d2.w = SM\_Creat function code  
                   d3.w = Coding Information  
                   d4.w = Number of sound groups

Output:           d0.w = Soundmap identifier  
                   (a0) = Soundmap address

Error Output:   cc    = Carry bit set to one  
                   d1.w = Error code

Possible Errors: E\$BPNuM, F\$SRqMem, E\$IllPrm, E\$IIdFull, E\$MemFul,  
 F\$SRtMem

Coding Information Format:



E = 1 - Emphasis ON  
 0 - Emphasis OFF

AL = Audio Level

0 - Level A mono	1 - Level A stereo
2 - Level B mono	3 - Level B stereo
4 - Level C mono	5 - Level C stereo



VII.2 File Managers

---

**Soundmap descriptor**

Offset	Length	Name	Description
0	2	SMD_Id	Id of the soundmap
2	2	SMD_StLoop	Start of Loop for loopback
4	2	SMD_EnLoop	End of Loop for loopback
6	2	SMD_LpCnt	Loopback count
8	2	SMD_LCntr	Counter for looping
10	4	SMD_SMAAddr	Address of soundmap itself
14	4	SMD_SMSize	Actual size of soundmap
18	4	SMD_CurAddr	Current address of playing soundmap
22	4	SMD_ASYBlk	Pointer to ASY block
26	2	SMD_NoGrps	Number of soundgroups in map
28	2	SMD_GrpNum	Group which started sector's play
30	1	SMD_Coding	Coding information byte as defined in Chapter IV.3.2.4
31	1	SMD_Res1	Reserved Byte
32	4	SMD_LpAddr	Start address of Loop
36	12	SMD_Res2	Reserved

**SM\_Out - Output Soundmap**

SM\_Out causes the data in the soundmap to be output to the audio processor. If another soundmap is being output, then that output is discontinued after the current sector (18 sound groups) has finished playing, and the new soundmap is started.

If ADPCM audio is being played from the disc, the soundmap will override the audio from the disc for the duration of the soundmap. When the soundmap is completed, the audio will resume. Playing a soundmap does not interfere with sectors being transferred to memory. If PCM audio is being played, the SS\_CDDA function is aborted before the soundmap is played.

SM\_Out is an asynchronous operation and therefore uses an asynchronous status block as described in Figure VII.9.

If the play of the soundmap is finished or an error occurs, then a signal is sent to the application depending on the contents of the ASY\_Sig field of the asynchronous status block. This also holds for the play of a soundmap which is discontinued by a SM\_Off, SM\_Close, SS\_CDDA, I\$Close or a new SM\_Out call. In these cases, the system will not set the error bit in the ASY\_Stat field.

Input:                   d0.w = Path number  
                          d1.w = SS\_SM setstat code  
                          d2.w = SM\_Out function code  
                          d3.w = Soundmap identifier  
                          (a0) = Pointer to asynchronous Status Block

Output:                   None  
Error Output:           cc    = Carry bit set to one  
                          d1.w = Error code

Possible Errors: E\$BPNuM, E\$UnID

**SM\_Off - Turn Off Soundmap Output**

SM\_Off causes output from the current soundmap to be discontinued and the audio processor to be turned off.

Input:           d0.w = Path number  
                  d1.w = SS\_SM setstat code  
                  d2.w = SM\_Off function code

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**SM\_Close - Close Soundmap**

SM\_Close de-allocates the memory associated with a soundmap.

Input:           d0.w = Path number  
                  d1.w = SS\_SM setstat code  
                  d2.w = M\_Close function code  
                  d3.w = Soundmap identifier

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, F\$SRtMem, E\$UnID

**SM\_Cncl - Conceal Error in Soundmap**

SM\_Cncl will perform the error concealment function provided by the audio driver.

Input:           d0.w = Path number  
                  d1.w = SS\_SM setstat code  
                  d2.w = SM\_Cncl function code  
                  d3.w = Soundmap identifier  
                  (a0) = Pointer to error information structure (see  
                          description of PCL\_Err under SS\_Play)

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNuM, E\$UnID

**SM\_Info - Return Pointer to Soundmap Descriptor**

SM\_Info returns pointer to the soundmap descriptor for information purposes.

Input:           d0.w = Path number  
                  d1.w = SS\_SM getstat code  
                  d2.w = SM\_Info function code  
                  d3.w = Soundmap identifier

Output:           (a0) = Pointer to Soundmap Descriptor

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNun, E\$UnID

**SM\_Stat - Return Soundmap Status**

SM\_Stat returns information about the soundmap currently being played. This call passes a buffer which is initialized by the driver.

Input:           d0.w = Path number  
                   d1.w = SS\_SM getstat code  
                   d2.w = SM\_Stat function code  
                   (a0) = pointer to soundmap status block

Output:           (a0) = pointer to soundmap status block

Error Output:    cc    = Carry bit set to one  
                   d1.w = Error code

Possible Errors: E\$BPNum, E\$NoPlay

## Format of soundmap status block

Offset	Length	Description
0	2	sector number currently being played
2	2	total sectors in soundmap
4	2	loops remaining for this sector
6	2	reserved

### 2.2.4.2 Sound Data Manipulation Functions

#### **SD\_MMix - Mix Monaural to Stereo**

The data from two monaural input soundmaps is mixed into stereo and placed in the output soundmap. The two input soundmaps must be of the same level. Mixing start point of input soundmap 1 is given by input parameter. The size of mixing is determined as the minimum size of three soundmaps.

Input:           d0.w = Path number  
                  d1.w = SS\_SD setstat code  
                  d2.w = SD\_MMix function code  
                  d3.w = Input soundmap 1 identifier  
                  d4.w = Input soundmap 2 identifier  
                  d5.w = Output soundmap identifier  
                  d6.w = Mix start sector number for input soundmap 1

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$UnID, E\$IIIPrm



**SD\_SMix - Mix Stereo to Stereo**

The left/right channel data of one channel from the first specified soundmap is mixed with the left/right channel data of one channel from the second specified soundmap and placed in the output soundmap. The two input soundmaps must be of the same level. In all cases, the data from the first soundmap is put in the left channel of the output soundmap and the data from the second soundmap is put in the right channel of the output soundmap. Mixing start point of input soundmap 1 is given by input parameter. The size of mixing is determined as the minimum size of three soundmaps.

Input:

- d0.w = Path number
- d1.w = SS\_SD setstat code
- d2.w = SD\_SMix function code
- d3.w = Input soundmap 1 identifier
- d4.w = Input soundmap 2 identifier
- d5.w = Output soundmap identifier
- d6.w = Mix start sector number for input soundmap 1
- d7.w = Left/right selection
  - 0 = Left channels of both soundmaps
  - 1 = Left channel of soundmap 1 and right channel of soundmap 2
  - 2 = Right channel of soundmap 1 and left channel of soundmap 2
  - 3 = Right channels of both soundmaps

Output: None

Error Output:

- cc = Carry bit set to one
- d1.w = Error code

Possible Errors: E\$BPNuM, E\$UnID, E\$IIlPrm

**SD\_Loop - Set Soundmap Loopback Points**

SD\_Loop sets the loopback points for a soundmap. If the loopback points are set when this soundmap is played (via SM\_Out), only the portion of the soundmap between the starting and ending sound groups is played. This portion will play the number of times specified by the loop count parameter.

If the soundmap is already being output when this call is made, output will continue until the audio processor reaches (or is past) the end sound group, at which time the loop count is decremented and the looping section is played again, if necessary.

The loopback points are specified in terms of sound groups but they will be rounded to sector boundaries (an integer multiple of 18 sound groups) such that the sector containing the starting sound group is played and the sector containing the end sound group is played.

Input:

- d0.w = Path number
- d1.w = SS\_SD setstat code
- d2.w = SS\_Loop function code
- d3.w = Soundmap identifier
- d4.w = Starting sound group number
- d5.w = Ending sound group number
- d6.w = Loop count

Output: None

Error Output:

- cc = Carry bit set to one
- d1.w = Error code

Possible Errors: E\$BPNum, E\$UnID, E\$IIIPrm

**2.2.4.3 Sound Control Functions****SC\_Atten - Set Attenuation**

The decoded left and right sound signals are both mixed into the final left and right output signals of the CD-I player. This essentially results in four signal paths, right to right, left to right, left to left, and right to left. SC\_Atten controls the amount of signal attenuation in each of these paths. This function's uses include control of the overall volume of sound, adjust balance for stereo sound, and to adjust the position of a sound in the stereo space.

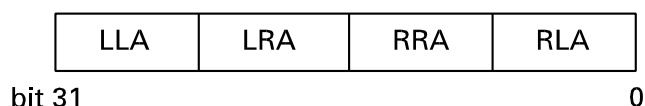
Input:           d0.w = Path number  
                   d1.w = SS\_SC setstat code  
                   d2.w = SC\_Atten function code  
                   d3.l = Attenuation values

Output:           None

Error Output:   cc    = Carry bit set to one  
                   d1.w = Error code

Possible Errors: E\$BPNum

Attenuation Values Format:



LLA = Left input to left output attenuation  
 LRA = Left input to right output attenuation  
 RRA = Right input to right output attenuation  
 RLA = Right input to left output attenuation

VII.2 File Managers

---

**2.2.5 Open and Close Service Requests**

This section specifies the `I$Open` and `I$Close` service requests used to open and close paths to CD files and the Audio Processor.

**2.2.5.1 I\$Open - Open Path to Specified Device/File**

This call is used to get all necessary information and set up all necessary data structures to be able to access a particular file or the Audio processor. If successful, a path number is returned which is used in subsequent I/O calls to identify the open file or device.

Input:           d0.b = Access mode  
                  (a0) = Pointer to path name

Output:           d0.w = Path number  
                  (a0) = Pointer to updated path name

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: `E$PthFul`, `E$BPNam`, `E$FNA`, `E$PNNF`, `E$Share`, `E$BNam`,  
`E$BTyp`, `E$DevBsy`, `F$PrsNam`, `F$SRqMem`, `E$BMode`

Access Mode Format:

Bit Number	Description (if set to one)
0	Read
1	Write (audio only)
2	Execute
3-5	Reserved
6	Non-shareable use
7	Directory

When opening files on the CD-I disc, there is a measure of file security provided for by CDFM. This is in the form of a file owner and an attribute field. Each file has a group/user ID that identifies the file's owner (see III.3.2 Owner ID/Attribute fields). The attribute field tells CDFM in which modes a file may be accessed. These fields indicate whether a file may be opened for reading or for execution by the owner, by anyone in the owner's group, or by the public. Public means any user with a different group ID.

## VII.2 File Managers

---

Whenever a file is opened, access permissions are checked on all directories specified in the pathlist, as well as the file itself. If the user does not have permission to read a directory, then the user may not read any files in that directory.

A super user (a user with a group ID of 0) may access any file in the system. Files containing modules that are owned by the super user must also be owned by the super user. If not, the modules contained within the file will not be loaded.

In all of these cases the "user's ID" is inherited from the process which spawned (forked) the application which is currently running. However, if the application primary module is owned by the super user, the application process may change its ID at any time (through F\$User) thus making a different set of "protected" files available for access.

### **Opening a Raw Device**

If a process has a super user ID it may open a device for Raw I/O. This allows the process to access the information on the device independently of the file structure. To do this, the process would append the "@" symbol to the end of the device name when executing I\$Open, e.g. "/cd@". This is the only mechanism for accessing structures such as the disc label.

VII.2 File Managers

---

**2.2.5.2 I\$Close - Close a File**

Close terminates the use of the I/O path. It causes the path number and all memory used to support the path to be returned to the system. If there is an asynchronous I/O function executing on this path the Close function aborts the asynchronous routine before returning to the application.

Input:           d0.w = Path number

Output:           None

Error Output:   cc    = Carry bit set to one  
                 d1.w = Error code

Possible Errors: E\$BPNum

## VII.2 File Managers

---

### 2.3 User Communications Manager (UCM)

#### 2.3.1 Introduction

This section describes the functions and capabilities of the User Communications Manager (UCM) and its associated drivers. The UCM provides the software interface to a CD-I player's video output device and user input devices. Content providers will use the UCM functions in their application programs to provide meaningful and interactive visual presentations to the users of CD-I players. The UCM provides the following functional capabilities:

Support for the CD-I system's visual effects and image mixing capabilities. Content providers have available to them well known motion picture techniques such as pans, wipes, fades, mattes, cuts, and dissolves.

Ability to create and maintain many separate images in memory and to copy and exchange parts of images in various ways. This allows applications to create a background image and images of separate foreground objects. The foreground images can be placed on the background image and even moved around, creating an animation effect.

A set of two dimensional graphics drawing functions for creating or modifying images. The drawing functions include line, arc, filled areas, and a powerful form of the bit block transfer.

High quality text in multiple fonts, sizes, and styles. Character sets for virtually all languages can be supported.

Functions for user input devices such as keyboards and mice are provided so that content providers may create interactive applications.

The User Communications Manager also provides hardware independent support for user input-output functions. It will not matter to an application whether a particular function is emulated in software or executed by a hardware coprocessor. The function set is also designed to be executed efficiently in a real-time environment yet be powerful enough to implement higher level functions.

VII.2 File Managers

---

UCM supports the following service requests:

Figure VII.17 **UCM Service Requests**

Name	Description
Open	Open path to specified device (see VII.2.3.3.1)
Read	Read characters from the keyboard (see VII.2.3.6)
Write/WriteLn	Write text to the display (see VII.2.3.4.8)
Getstat	Get specified status information
Setstat	Set specified status information
Close	Close path to specified device (see VII.2.3.3.2)

For a more detailed description see A VII.1.

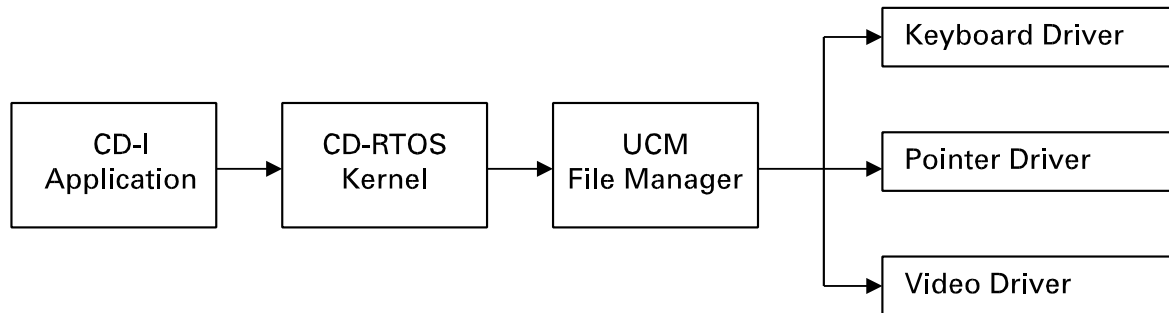
### 2.3.2 Features and Concepts

This section of the specification describes the major features and basic concepts upon which the User Communications subsystem is built.

#### 2.3.2.1 Environment

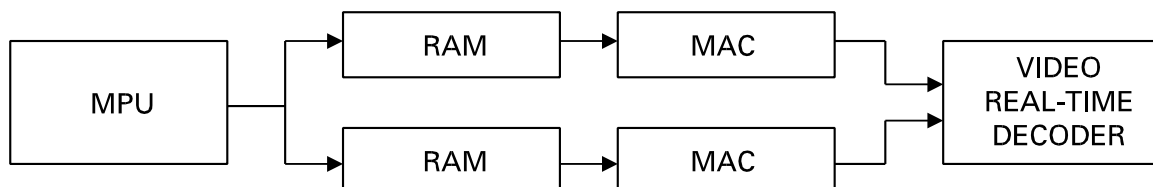
The User Communications Manager conforms to the standard input-output architecture used in CD-RTOS. The diagram in Figure VII.18 shows how the different parts (or levels) of the system are related to each other and the application. Any input-output subsystem within CD-RTOS is implemented on three levels. On the top level is the CD-RTOS kernel. It defines the application interface to the input-output system. The lowest level is the driver which interfaces to the actual hardware involved. The drivers implement a set of functions which are designed for a particular class of hardware device. For example, drivers for Random Block File Manager (for disks) have functions which read and write sectors and format the disk (see Appendix VII.1). Drivers for the Sequential Character File Manager (for CRT terminals) read and write characters. The file manager is at the intermediate level. One of its responsibilities is to act as a translator between kernel functions and driver functions. The kernel defines the input-output functions to be the same for all classes of devices whereas the driver does not. The file manager also performs any house-keeping duties which are required for its class of device.



Figure VII.18 **Functional Organization of the User Communication Manager**

### 2.3.2.2 Display Architecture

An example of the video section of a CD-I player is shown in Figure VII.19. There are three entities which perform the graphics functions, a drawing processor, a memory access controller, and a video real-time decoder. The drawing processor (whose functions are performed by the MPU in the example) executes all of the drawing and text functions and can be implemented in hardware or emulated in software. It is also responsible for the creation and maintenance of images in the player's memory (RAM). The memory access controller (MAC) retrieves pixel data from the memory and passes it to the video real-time decoder for display. The CD-I system requires that two memory access controllers be used. Each memory access controller processes a single image plane. The video real-time decoder takes the pixel data given to it by the MACs and converts it to its representative electrical signals. For more details see Chapter V.4.1.

Figure VII.19 **Example of the Video Section of a CD-I Player**

## VII.2 File Managers

---

### 2.3.2.3 Drawmaps

The primary data object for the UCM and video driver is the draw-map. Drawmaps are used to store images. Applications may draw on drawmaps using the drawing functions. A drawmap may be virtually any size, limited only by the memory space available.

The drawmap has two formats, the rectangular array format and the list format. In the array format, each element of the array represents a single pixel. The array is two dimensional and each element of the array corresponds to a single pixel. The position of the pixel within the array also corresponds to its position on the display screen. Each line of pixels in the array must be an integral number of long words in length.

The list format is provided to support run-length coding. In the list format drawmap, each element represents a sequence of pixels which have the same color or pixel value. Since there is no one-to-one correspondence between a pixel coordinate and its location in the list, graphics drawing functions can not be used on a list format drawmap. Each line of pixels is represented in the list by an integral number of bytes.

Since the CD-I video hardware allows pixels to have different representations, drawmaps have a data type. The allowable data types are four-bit CLUT, seven-bit CLUT, eight-bit CLUT, RGB 555, DYUV and (for the extended case) DYUV + QHY. Since an individual pixel's color value depends on the values of all previous pixels in the line, the graphics drawing functions may not be used on a DYUV type drawmap. The DYUV type drawmap may, however, be manipulated using the drawmap control functions.

When an image comprises two or more differently coded subscreens, a separate drawmap should be defined for each subscreen.

Every drawmap also has associated with it a line pointer table. This is a linear array of pointers to the beginning of each line in the drawmap. An RGB 555 drawmap occupies two blocks of memory and therefore requires two line pointer tables. List format drawmaps also have line pointer tables, but only the first entry is actually initialized by the driver. The line pointer table may be used to eliminate the need for copying lines of the drawmap around when an error in video data needs to be concealed. It also helps simplify the setup of a display control program for displaying the drawmap.

The drawing functions may be used on List Format (RL3 and RL7) drawmaps provided that the application has initialized the line pointer table. The application should be aware that unpredictable results may occur.

#### 2.3.2.4 Display Control Program

Drawmaps and their associated functions are concerned only with the management and manipulation of images in memory. The display control program (DCP) determines how those images will appear on the display screen. Application programs use display control programs to perform functions such as matte, wipe, or scroll. The display control program is essentially a set of instructions which are performed by the video display hardware when it is not busy retrieving and displaying pixel data (i.e. during vertical and horizontal retrace periods).

There are two Display Control Programs, one for each display path, operating in conjunction. A display control program has two major components, the field control table (FCT) and the line control table (LCT). The field control table is a one dimensional array of instructions which are interpreted by the display hardware during every vertical retrace period. The maximum length of the field control table, available to the application, is 1024 instructions.

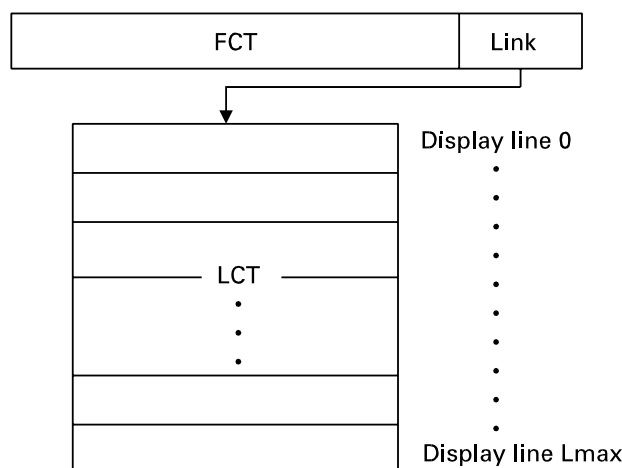
The line control table is a two dimensional array of instructions. Successive lines of instructions in the line control table are interpreted by the display hardware during successive horizontal retrace periods prior to the display of lines of pixel data. The maximum number of instructions that may be interpreted during a horizontal retrace period is eight in a Base Case CD-I system. The maximum number of lines in the line control table is 512 in normal resolution and 1024 in high resolution. An LCT is not required, but if one is used it has a minimum size requirement of two lines.

For RGB 555, which occupies both display paths, two LCTs and two FCTs are needed. These must be independently created, in separate planes, and written to.

The UCM provides many functions for manipulating field and line control tables. For field control tables, functions are provided for reading and writing any number of instructions. For line control tables, functions are provided for reading and writing rectangular arrays of any size and single instructions. The application is also able to create an arbitrary number of these tables, load them with instructions, and rapidly switch between them for high speed special effects.

An example of a DCP and the sequence of its interpretation is shown in Figure VII.20. At the beginning of the vertical retrace period, the display control hardware interprets the instructions in the selected FCT. Interpretation of the FCT is completed before the start of the active display period.

Figure VII.20 Example of a DCP and its Sequence of Interpretation



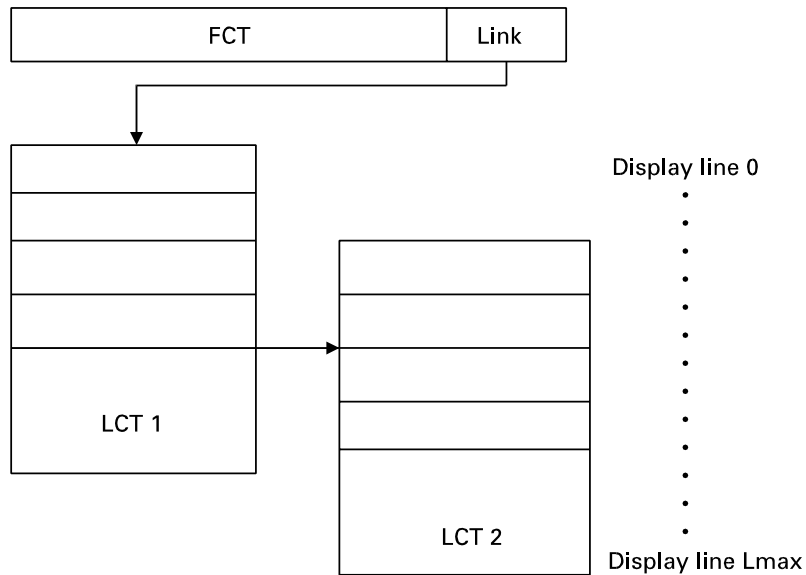
Contained within the field control table is a link to a line control table (set by the DC\_FLnk function). This link can point to any line of the line control table, not just to the beginning. Prior to the display of the first line of pixels, the display control hardware will interpret the instructions contained in the line of the LCT pointed to by this link. For this line and this line only, the system cannot guarantee to execute the instructions in the first linked line of the LCT for plane A and the instructions in the first linked line of the LCT for plane B synchronously, i.e. one plane's LCT line may be executed before the other plane's FCT has been executed. For 625 and 525 line systems image interchange, the application can create a line control table whose length is equal to the number of display lines (see A.V.2.2.2).

In the simplest case the display control hardware continues to synchronously interpret successive lines of the LCT, along with its associated video data, until the last line of the display is completed. The interpretation process is then repeated for the next display field. For interlaced scan high resolution the two halves of the LCT are interpreted in alternate fields.

A display control program may also have several line control tables which can be linked together in a manner similar to the way an FCT links to an LCT. An example of a multiple LCT display control program is shown in Figure VII.21. When the display control hardware sees a link in an LCT, it will interpret instructions beginning with the line in the LCT pointed to by the link at the next horizontal retrace period. Multiple LCT's in a DCP may be used for the construction of subscreens on the display. By manipulation of the links between LCT's, scrolling in subscreens is easily performed.

**Note:** It is not allowed to link one LCT to another LCT from a line in the first LCT linked to by the FCT for the given plane.

Figure VII.21 **Example of a Multiple LCT Display Control Program**



**Video Events**

A DCP instruction is available to provide synchronization to Display Scanning (see V.5.6). The video interrupt which occurs when the selected display line is reached can be detected using the CD-RTOS F\$Event system call or the DC\_SSig system call.

When the video driver is initialized it creates an event named "line\_event". The driver, when a video interrupt occurs, pulses the event to the value one using Ev\$Pulse. To access the event an application must link to it using the Ev\$Link function. This function returns an event identifier which is used as a parameter to the Ev\$Wait and Ev\$WaitR functions to wait for the video interrupt event. Note that Ev\$Read cannot be used since the driver uses the Ev\$Pulse function. When finished with the event, the application should unlink it using Ev\$UnLnk. For more information see A VII.1.

## VII.2 File Managers

---

\*\*\*\*\* EXTENSION \*\*\*\*\*

### High Resolution

High resolution images may be displayed either in interlaced or line-sequential mode. To ensure that any high resolution image may be displayed in either mode without unacceptable constraints or overheads it is **recommended** that the application should:

- 1) Create a drawmap with even and odd lines separated. This organization is suitable for either scan mode.
- 2) Create one high resolution FCT which, for interlaced scan mode, may need to be duplicated by the video driver.
- 3) Create one high resolution LCT which, for interlaced scan displays, will be physically organized with odd/even lines separated.
- 4) Write display start address instructions in lines 0 and 1 of the LCT for each display field.

**Note:** If the application creates a drawmap and LCT organized for line-sequential display, explicit line start pointers must be inserted in every line of the LCT. This is to allow interlace display.

\*\*\*\*\*

### DCP Instructions

The instructions which are contained in the display control programs are all 32-bit long words<sup>12</sup>. These instructions consist of an op code followed by a 24-bit parameter. A summary of the available instructions is provided in Figures VII.16-18 along with explanations of restrictions on their use. Complete descriptions of these commands can be found in Chapter V. Figure VII.22 describes the control table entries available for plane 0 only, Figure VII.23 describes the control table entries available for plane 1 only, and Figure VII.24 describes the control table entries available for both planes.

---

<sup>12</sup> Since the base case supports a 16 bit data bus version of the CPU (see VIII.8.2) the writing of a 32-bit DCP instruction is done in 2 cycles. This can lead to unexpected results if a partially updated instruction is executed by the display logic. The application can avoid this situation by double buffering or by synchronizing the write to the display scanning.

VII.2 File Managers

---

The application is required to provide a display control program for each image plane. These two display control programs are interpreted concurrently by the display control hardware. The instructions specific to each plane are controlled only by the display control program in that plane. Most global control instructions must be placed in the display control program for path 0, however, some may be used in both planes.

Figure VII.22

**Control Program Instructions for Path 0 only**

Code	Action	Parameter
\$C0	Select image coding methods	V 4.6.1
\$C1	Load transparency control information	V 5.7.3
\$C2	Load plane order	V 5.7.1
\$C4	Load transparent color for plane A	V 5.7.2.2
\$C7	Load mask color for plane A	V 5.7.2.2
\$CA	Load DYUV start value for plane A	V 4.6.2
\$D8	Load backdrop color	V 5.13
\$D9	Load mosaic pixel hold factor for A	V 5.11.1.1
\$DB	Load image contribution factor for A	V 5.9

Figure VII.23

**Control Program Instructions for Path 1 only**

Code	Action	Parameter
\$C6	Load transparent color for plane B	V 5.7.2.2
\$C9	Load mask color for plane B	V 5.7.2.2
\$CB	Load DYUV start value for plane B	V 4.6.2
\$DA	Load mosaic pixel hold factor for B	V 5.11.1.1
\$DC	Load image contribution factor for B	V 5.9

Figure VII.24

**Control Program Instructions for both Planes**

Code	Action	Parameter
\$10	No operation *	-
\$20	Load LCT pointer *	Note 1
\$40	Load display line start pointer *	V 4.5.2.2
\$60	Signal when scan reaches this line *	V 5.6
\$78	Load display parameters *	V 4.6.1
\$80-BF	Load CLUT color 0-63 (of current bank)	V 5.5
\$C3	Set CLUT bank *	V 5.5
\$DO-D7	Load matte register 0-7	V 5.10.3
\$80-87	Load QHY level 0-7	V 4.4.3.4
\$C3	Set QHY bank	V 4.4.3.4

\* = This action may be set independently in each path.

**Note 1:** This instruction may only be written by means of the UCM calls DC\_LLnk and DC\_FLnk. It is the responsibility of the application to insure that space is left in the LCT or FCT by means of this mechanism before executing the DC\_LLnk or DC\_FLnk commands.

The video driver will place this instruction in the last column of the LCT when using the DC\_LLnk command. In the case of DC\_FLnk, the video driver will place this instruction beyond the end of the application FCT.



### 2.3.2.5 Coordinate Transformation

The User Communications Manager uses a variation of the Cartesian Coordinate System to address pixels within a drawmap. The UCM's coordinate system differs from Cartesian coordinates in that vertical coordinates increase in value from top to bottom, rather than from bottom to top. Horizontal coordinates, as in the Cartesian system, increase in value from left to right. The UCM also limits the range of coordinates. The minimum possible coordinate is -32768 and the maximum is 32767. The origin (point 0,0) is also variable for each drawmap. The origin can be set at the top-left corner of the drawmap, the center of the drawmap, or even at the outside of the drawmap. The UCM also differs from the Cartesian system in that the precision of coordinates is not infinite. Coordinates for UCM are restricted to integer values.

To maintain resolution independence, application programs need to 'see' a single coordinate space for all drawmaps. In other words, a circle should appear to be the same size on the display screen whether it is drawn in a normal, double, or high resolution drawmap. The UCM fills this need by having applications treat all drawmaps as if they were high resolution drawmaps. If the drawmap is not a high resolution drawmap then the coordinates specified by the application are scaled according to the actual resolution of the drawmap.

Consequently, coordinate values as well as dimensions (height, width) are always expressed according to High Resolution conventions in this chapter except when otherwise mentioned.

### 2.3.2.6 Regions

Regions divide the set of points in the UCM coordinate space into two disjoint sets of points. Their main uses are for graphics drawing and construction of mattes. When drawing, regions are used to limit the area of drawing in a drawmap. The shapes specified by a region can also be drawn. The data in the region data structure is also suitable for use with the setup of mattes using the display control program functions.

Regions are created from the basic shapes of rectangle, elliptical cornered rectangle, polygon, circle, circular wedge, ellipse, and elliptical wedge. More complex shapes are created from these basic shapes using intersection, union, exclusive-or, and difference functions. Regions may be any shape or size, however, the region data structure itself is limited to 65536 bytes in length.

VII.2 File Managers

---

The region data structure has three parts, the region header, the line offset table, and the transition data table. The region header contains general information about the region. The line offset table is a list of offsets to each line of transition data in the transition data table. The transition data table has lists of horizontal points which represent the transition points in and out of the region for each line in the region. The data in the region is in UCM coordinate form.

The format of the region header is specified below.

Figure VII.25 **Region Header Format**

Offset	Length	Description
0	2	Region identifier
2	1	Region type (see VII.2.3.4.3)
3	3	Reserved
6	2	Length (in bytes) of region data structure
8	2	Horizontal coordinate of upper left corner of boundary rectangle
10	2	Horizontal coordinate of lower right corner of boundary rectangle
12	2	Vertical coordinate of upper left corner of boundary rectangle
14	2	Vertical coordinate of lower right corner of boundary rectangle

The line offset table, which directly follows the region header, is a list of unsigned 16-bit integers which have the byte offset from the beginning of the region data structure to the transition data in the transition data table for that line. The first entry in the table points to the transition data for the first line in the region. The vertical coordinate of the upper left corner of the region's boundary rectangle is the first line in the region. If there is no transition data for a line in a region, then the corresponding entry in the line offset table is zero.

The transition data table is a list of lists and immediately follows the line offset table. There is one sublist for each line in the region for which there is transition data. The first 16-bit word in each sublist contains the number of transition pairs in that sublist. Each transition pair that follows is a pair of 16-bit words. The first word in the pair is the horizontal coordinate of the transition into the region and the second word in the pair is the horizontal coordinate of the transition out of the region. Both transition points are considered to be in the region.

## VII.2 File Managers

---

### 2.3.2.7 Graphics Drawing

Basic drawing functions such as line, rectangle, circle, and arc are provided by the UCM and video driver. It is also possible to draw these elements using different sizes and patterns of lines. Arbitrary bounded regions of drawmaps may also be filled using a pattern. A form of the bit block transfer is also provided. All of the drawing primitives also perform logical or arithmetic operations between the drawing data and the drawmap data.

### 2.3.2.8 Pattern Indirect Drawing

Pattern Indirect Drawing is the method used by graphics drawing operations to transfer pixel data to the drawmap. The pattern is a sixteen by sixteen pixel bitmap. Like drawmaps, patterns also have a data type. The allowable data types are the same as those for drawmaps except that DYUV is not allowed. Patterns can also have single-bit, double-bit, and quad-bit data types. When these data types are used, the pixels are expanded using a set of color registers to obtain the actual data that will be drawn in the drawmap. The pattern's pixel value selects which color register the drawing data is obtained from. This compression technique is also very useful for fonts, which could take up a large amount of space if actual pixel data were to be used.

To determine which pixel in a pattern is used to draw a specific pixel in a drawmap, a correspondence is made between the pixels in a drawmap and the pixels in a pattern. Since a drawmap is usually larger than a pattern, many pixels in the drawmap correspond to a single pixel in a pattern. In most cases, pixel (0,0) in the drawmap correspond to pixel (0,0) in the pattern. Pixels (16,0), (32,0), and (48,0) also correspond to pixel (0,0) of the pattern. Likewise, pixels (0,16), (0,32), and (0,48) also correspond to pixel (0,0) of the pattern. This correspondence, which is called alignment, is also adjustable. For example, pixel (0,0) of the drawmap can be made to correspond to pixel (1,1) of the pattern.

### 2.3.2.9 Character Output Encoding

Due to the international use of CD-I, the UCM and the video driver do not support any one character code standard for their text output functions. They only provide the basic mechanisms so that all character code standards may be supported. The UCM and the video driver only use character codes to select glyphs from a font module. A glyph is the bitmap image of a symbol. For example, the bitmap image for the letter 'a' is a glyph for the letter 'a'.

## VII.2 File Managers

---

All of the glyphs in a font module are numbered using an unsigned sixteen bit integer. This glyph number is required to access a glyph in a font. Since some character code standards have codes which are eight bit quantities and some have codes which are sixteen bit quantities, the UCM provides three methods of interpreting (e.g. translating to glyph numbers) the character data passed to the text functions, eight bit, seven/fifteen bit, and sixteen bit. The eight bit method is usable for most alphanumeric type languages. When the eight bit method is being used only those glyphs in a font which are numbered from 0 to 255 are accessible. In the seven/fifteen method, the most significant bit of a character code is checked to determine whether the code is a seven bit quantity (one byte) or a fifteen bit quantity (two byte). When the most significant bit is reset (i.e. to zero), then a seven bit quantity is represented. All seven bit quantities map to glyph numbers 0 to 127 and all fifteen bit quantities map to glyph numbers 32768 to 65535. This method is useful because it can support ISO 646 (or ISO 8859-1: \$00 - \$7F) code and shifted JIS Kanji code in the same data stream. In the sixteen bit mode, all character codes map directly to glyph numbers. The text output functions are also able to select glyphs from up to four different fonts based on the character code passed to the function. Each font supports a subrange of the possible range of glyph numbers. Since all character codes map uniquely to glyph numbers, each font, in effect, supports a subrange of character codes. The glyph is selected from the font that supports the required subrange. If two or more fonts contain glyphs with the same glyph number(s), the font with the lowest font number will be used.

### 2.3.2.10 Text Fonts

Text in virtually any size, style, or font is supported by the software. These attributes are controlled by a font module, which is a CD-RTOS data module that contains the bitmap images for each displayable character.

A font module consists of six parts, a module header, the font data section, the glyph offset table, the glyph data table, the font bitmap, and the module CRC. The module header and module CRC are standard parts of a CD-RTOS memory module and are defined in Appendix VII.1.

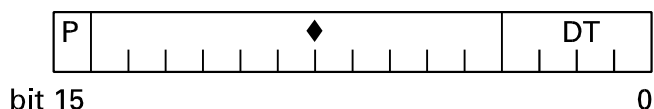
At the module's entry point (which is declared in the module header) is the font data section. The font data section is usually placed immediately following the module header and contains general information about the font. The format of the font data section is as follows.

## VII.2 File Managers

Figure VII.26 Structure of Text Font Data Section

Offset	Length	Name	Description
0	2	fnt_type,	Font type & flags
2	2	fnt_width,	Maximum Glyph cell width <sup>13</sup>
4	2	fnt_height,	Glyph cell height <sup>13</sup> (ascent+descent
6	2	fnt_ascent,	Ascent <sup>13</sup> of character cell above baseline
8	2	fnt_descent,	Descent <sup>13</sup> of character cell below baseline
10	2	fnt_pxlsz,	Pixel size in bits
12	2	fnt_frstch,	First character value of font
14	2	fnt_lastch,	Last character value of font
16	4	fnt_lflen,	Line length of first font bitmap in bytes <sup>15</sup>
20	4	fnt_offstbl,	Offset <sup>14</sup> to glyph offset table
24	4	fnt_databl,	Offset <sup>14</sup> to glyph data table
28	4	fnt_map1off,	Offset <sup>14</sup> to first bitmap <sup>15</sup>
32	4	fnt_map2off,	Offset <sup>14</sup> to second bitmap <sup>15</sup> RGB only

Font Data Type/Flags Format:



- P = Proportional/mono-spaced flag
- 0 = Monospace
- 1 = Proportional spaced
- DT = Data type
- 0 = Single-bit
- 1 = Double-bit
- 2 = Quad-bit
- 3 = CLUT4
- 4 = CLUT7
- 5 = CLUT8
- 6-8 = Reserved
- 9 = RGB555
- 10-15 = Reserved

<sup>13</sup> Values in pixels.

<sup>14</sup> Offset values are taken from the beginning of the font data section.

<sup>15</sup> fnt\_map1off and fnt\_map2off must reference addresses which are long word aligned. fnt\_lflen must be divisible by 4.

VII.2 File Managers

---

The glyph offset table is usually placed immediately following the font data section. This table is required in all fonts. It is merely a one dimensional array of sixteen bit integers which represent the horizontal distance in pixels from the left edge of the font bitmap to the left edge of the glyph. There must be one entry in the glyph offset table for each glyph in the range specified by the first glyph number and last glyph number in the font data section. If a particular glyph number in that range is not displayable, then its entry in the glyph offset table should be set to -1.

The glyph data table is usually placed immediately following the glyph offset table. There must be one entry in the glyph data table for each glyph in the range specified by the font data section, whether displayable or not. Each entry in the glyph data table is formatted as shown in Figure VII.27.

Figure VII.27 **Structure of each Entry in the Glyph Data Table**

Offset	Length	Description
0	1	Character width in pixels
1	3	Reserved

The font bitmap is usually placed immediately following the glyph data table. The font bitmap is a rectangular array of pixels of the same form as a drawmap. Its height is the same as the character cell height and its width is equal to the sum of all of the widths of the glyphs in the font. All of the glyphs for the font are placed in the font bitmap in sequential order from left to right.

When a glyph is drawn, the pixel data is used in a similar way to the pixel data in a pattern when used for graphics drawing (VII.2.3.2.8). Fonts have the same data types as patterns. If a font has single, double or quad-bit data type, each pixel is expanded to color data via the color registers. For single-bit pixels only entries 0 and 1 of the color registers are used. Quad-bit pixels can use all 16 entries.

If the font contains actual color data then that data is transferred directly to the drawmap, so long as the data types of the font and the drawmap are the same.

VII.2 File Managers

---

**2.3.3 General UCM Service Requests**

This section specifies the service requests and functions which apply to all UCM devices.

By function, a list of error codes returned is given. The system calls (e.g. F\$SRqCMem) used by some of those functions are also indicated: in this case all errors returned by the system calls are also possible (see Appendix VII.1).

**2.3.3.1 \$Open - Open Path to Specified UCM Device**

This call is used to get all necessary information and set up all necessary data structures to be able to access a particular device. If successful, a path number is returned which is used in subsequent I/O calls to identify the open device. If the device to be opened is not available an error is returned. If two paths are opened to the same device, function calls to the different paths have identical effects (e.g. a drawmap can be created through one path and then closed through the other).

Input:           d0.b = Access mode  
                  (a0) = Pointer to pathname of device

Output:          d0.w = Path number  
                  (a0) = Pointer to updated pathname of device

Error Output:   cc     = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$BPNam, E\$MNF, E\$Share, E\$BMode,  
F\$SRqMem, I\$Attach, I\$Detach

Access Mode Format:

Bit Number	Description (if set to one)
0	Read
1	Write
2-5	Reserved
6	Non-shareable use
7	Reserved

## VII.2 File Managers

---

### 2.3.3.2 **I\$Close - Close a UCM Device**

Close terminates the use of the I/O path. It causes the path number and all memory used to support the path to be returned to the system. If there is an asynchronous I/O function executing on this path the Close function waits until completion of the asynchronous routine before returning to the application.

Input:               d0.w = Path number

Output:             None

Error Output:     cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: F\$SRtMem, E\$BPNuM, F\$IRO, F\$Event, F\$UnLink,  
                  I\$Detach, E\$UnID

### 2.3.3.3 **SS\_SLink - Link Subroutine Module**

This function is used as a generalized method of expanding the command set of UCM. It utilizes a special subroutine module which is linked into UCM. When UCM receives an unknown Getstat or Setstat service request, it will pass the call to the subroutine module. If it is unknown to the subroutine module, the subroutine module should then pass the call to the driver.

There are four significant entry points into this external Subroutine Module: Init, Setstat, Getstat and Close.

Init will be called when the module is first linked to by UCM. It is provided to allow the module to initialize data structures and variables. Likewise, the Close entry point will be called when the path to UCM is closed. At this time the subroutine module should free any memory which was allocated during its execution.

The Setstat and Getstat entry points are provided to actually do the main work of the subroutine module.

The mechanism for this dispatching should be similar to the one currently used by CD-RTOS file managers and drivers. The execution entry point of the module points to the primary jump table. This jump table contains the relative addresses for the Init, SetStat, GetStat and Close routines (in that order). These offsets are two byte entries.



## VII.2 File Managers

---

When UCM wishes to pass control to one of these functions it will retrieve the appropriate address from the jump table and jump through to that code.

Space is reserved in the UCM device static storage for this subroutine module to use. Eighty bytes are available beginning at the offset V\_EXMOD.

**CAUTION:** Because this space is global to all paths open to UCM, it is impossible for two different subroutine modules to be linked at the same time. This applies even when they are linked by different processes. However, more than one process may be linked to the same subroutine module.

For more information, see A VII.1

**Input:**

- d0.w = Path number
- d1.w = SS\_SLink setstat code
- (a0) = Pointer to name of file manager subroutine module

**Output:** None

**Error Output:**

- cc = Carry bit set to one
- d1.w = Error Code

**Possible Errors:** E\$KwnMod, F\$Link, E\$BPNum

## VII.2 File Managers

---

### 2.3.4 Video and Graphics Functions

All of the functions defined in this section control what appears on the display. Throughout this section many parameters, such as the end point of a line, have both a horizontal and vertical component. Whenever a notation such as S:D (Source:Destination for identifiers), H:V (Horizontal:Vertical for coordinates) or W:H (Width:Height for sizes) is used, the source, horizontal or width component will be placed in the high-order word of the register or memory location and the destination, vertical or height component in the low-order word.

#### 2.3.4.1 Drawmap Control Functions

Virtually any number of drawmaps (up to the limit of memory) may be allocated. Functions are also provided to copy or exchange data between drawmaps.

The following functions are not allowed with RunLength drawmaps:

- DM\_Copy
- DM\_Exch
- DM\_TCpy
- DM\_TExc
- DM\_Read
- DM\_Write
- DM\_IrWr
- DM\_RdPix
- DM\_WrPix
- DM\_Cncl

**DM\_Creat - Create Drawmap**

DM\_Creat creates a drawmap comprised of one or two blocks of memory used to store and/or display images. The allowable data types are CLUT 4, CLUT 7, CLUT 8, RGB 555, DYUV, DYUV + QHY (extended case), RL3 and RL7. The size of the memory block(s) allocated depends on the two-dimensional size specified in d5.l and on the length specified in d6.l. The video driver calculates the amount of memory required by the specified size and selects the greater of the calculated amount and the specified length.

**Note:** When calculating the size of the drawmap, the driver first normalizes the width and height parameters as per the resolution specified, then compares that with the length parameter. Width and height may not be 0 (or 1 in normal resolution).

In the case of RL3 and RL7 data types, only the length parameter is used in calculating the size of the memory blocks. However, the height parameter is still used to calculate the size of the associated line pointer table, and must be greater than 0 (or 1 in normal/double resolution). In this case only, width is ignored and may be 0.

In the case of RGB 555, which creates two blocks of memory, the driver calculates the amount of memory required for each block and creates two blocks of that size.

In the case of DYUV + QHY, (extended case) the size and length are used to create a standard DYUV drawmap block in one plane. The length parameter in d7.l specifies the length of a QHY drawmap block in the other plane.

When video is transferred from a Real Time Record directly to the drawmap, the data is transferred one sector at a time. When this occurs, the size of the drawmap must be a multiple of 2324 bytes. When the "lines separated" flag is used, if the total size of the drawmap is  $n$ , the beginning of the second line starts at  $n/2$ .

The plane number specifies the memory bank (plane 0 or plane 1) that the memory block is allocated from. It is possible to allocate a drawmap in a plane that will not be able to display it (i.e. CLUT 8 in plane B). In the case of RGB 555, the plane parameter is ignored and the driver creates one memory block in each plane.

VII.2 File Managers

---

For a list of the permissible display combinations see section V.4.4.8. The DM\_Creat function returns the drawmap identifier and the address of the drawmap descriptor (this should be used for information purposes only).

Input:           d0.w = Path number  
                  d1.w = SS\_DM setstat code  
                  d2.w = DM\_Creat function code  
                  d3.w = Plane number  
                  d4.w = Data type/flags  
                  d5.l = Size (W:H) in UCM coordinates  
                  d6.l = Length in bytes  
                  d7.l = Length in bytes of QHY

Output:           d0.w = Drawmap identifier  
                  (a0) = Pointer to drawmap descriptor

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIIPrm, E\$BadMod, E\$IIdFull, F\$SRqCMem, F\$SRtMem, E\$BPNum

## VII.2 File Managers

**Drawmap Descriptor Format:**

<b>Offset</b>	<b>Length</b>	<b>Name</b>	<b>Description</b>
0	2	MD_DNum	Drawmap identifier
2	2	MD_DType	Data type and flags
4	4	MD_Mapl	Pointer to primary drawmap memory
8	4	MD_LnATbl1	Pointer to primary line pointer table
12	4	MD_MapSz1	Size of primary drawmap in bytes
16	2	MD_Plane	Plane number
18	2	MD_PxlSz	Size of a pixel in bits
20	2	MD_PMapW	Physical width of drawmap in pixels
22	2	MD_PMapH	Physical height of drawmap in pixels
24	2	MD_XOrg	Origin offset in X direction in UCM coordinates
26	2	MD_YOrg	Origin offset in Y direction in UCM coordinates
28	2	MD_CMinX	Minimum X of default clipping region in pixels
30	2	MD_CMaxX	Maximum X of default clipping region in pixels
32	2	MD_CMinY	Minimum Y of default clipping region in pixels
34	2	MD_CMaxY	Maximum Y of default clipping region in pixels
36	2	MD_PenW	Pen width in pixels minus one
38	2	MD_PenH	Pen height in pixels minus one
40	4	MD_PSStr	Pen style bit string
44	2	MD_PSCnt	Pen style repeat count
46	2	MD_TCol1	Transparency color for primary drawmap
48	4	MD_Patt	Pointer to current pattern
52	2	MD_PMode	Pattern mode
54	2	MD_POffX	X offset to pattern bit point in UCM coordinates
56	2	MD_POffY	Y offset to pattern bit point in UCM coordinates
58	32	MD_CRTbl	Color register table
90	2	MD_MpMthd	Mapping method for graphics text
92	2	MD_AFBeg0	Beginning glyph value for 1st active font for graphics text
94	2	MD_AFEnd0	Ending glyph value for 1st active font for graphics text
96	4	MD_AFData0	Pointer to beginning of first active font data for graphics text

(continued)

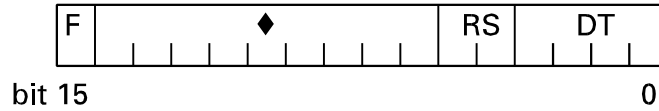
**Drawmap Descriptor Format** (continued)

<b>Offset</b>	<b>Length</b>	<b>Name</b>	<b>Description</b>
100	2	MD_AFBeg1	Beginning glyph value for 2nd active font for graphics text
102	2	MD_AFEnd1	Ending glyph value for 2nd active font for graphics text
104	4	MD_AFData1	Pointer to beginning of 2nd active font data for graphics text
108	2	MD_AFBeg2	Beginning glyph value for 3rd active font for graphics text
110	2	MD_AFEnd2	Ending glyph value for 3rd active font for graphics text
112	4	MD_AFData2	Pointer to beginning of 3rd active font data for graphics text
116	2	MD_AFBeg3	Beginning glyph value for 4th active font for graphics text
118	2	MD_AFEnd3	Ending glyph value for 4th active font for graphics text
120	4	MD_AFData3	Pointer to beginning of 4th active font data for graphics text
124	2	MD_RgnId	Region identifier of user clipping region
126	4	MD_Map2	Pointer to secondary drawmap memory
130	4	MD_LnATbl2	Pointer to secondary line pointer table
134	4	MD_MapSz2	Size of secondary drawmap in bytes
138	2	MD_TCol2	Transparency color for secondary drawmap
140	116	Reserved	

VII.2 File Managers

---

**Drawmap Data Type Format**



- F = Line pointer table format
- 0 = Even/odd lines interleaved
- 1 = Even/odd lines separated\*
- RS = Resolution
- 0 = Normal resolution
- 1 = Double resolution
- 2 = Reserved
- 3 = High resolution (extended case)
- DT = Data type
- 0-2 = Reserved
- 3 = CLUT4                      8 = DYUV
- 4 = CLUT7                     9 = RGB555
- 5 = CLUT8                    10 = Reserved
- 6 = RL3                        11 = DYUV+QHY
- 7 = RL7                        12-15 = Reserved

\* Even/odd lines separated is intended for error concealment and high resolution display. Separation of the two fields facilitates interlaced display of high resolution images without inhibiting the use of line-sequential scanning.

For graphics drawing UCM uses the line pointer table to locate lines.

**Resolution/Data Type Combinations/Restrictions**

Figure VII.28 **Resolution/Data Type Combinations/Restrictions:**

	Normal	Double	High
CLUT4	-	B	B
CLUT7	B	-	E
CLUT8	B	-	E
RL3	-	B	B
RL7	B	-	E
DYUV	B	-	E
DYUV+QHY	-	-	E
RGB555	B	-	-

- = Not specified
- B = Base Case
- E = Extended Case

**DM\_Org - Set Drawing Origin**

DM\_Org changes the position of the drawmap in the coordinate space. The default origin when the drawmap is created is at the upper left hand corner of the drawmap. The origin is specified as an offset from the top left corner of the drawmap.

Input:           d0.w = Path number  
                  d1.w = SS\_DM setstat code  
                  d2.w = DM\_Org function code  
                  d3.w = Drawmap identifier  
                  d4.l = Origin offset (H:V) in UCM coordinates

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$BPNum



**DM\_Copy - Copy Drawmap to Drawmap**

DM\_Copy copies a rectangular shaped area from one drawmap to another. The two drawmaps must have the same data type. The source drawmap identifier is placed in the high-order word of register d3 and the destination drawmap identifier is placed in the low-order word. If the specified size would result in data being copied from outside the source drawmap or to the outside of the destination drawmap, the size is reduced accordingly.

Input:           d0.w = Path number  
                  d1.w = SS\_DM setstat code  
                  d2.w = DM\_Copy function code  
                  d3.l = Drawmap identifiers (S:D)  
                  d4.l = Destination (H:V) coordinate  
                  d5.l = Source (H:V) coordinate  
                  d6.l = Size (W:H) in UCM coordinates

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BadMod, E\$UnID, E\$BPNuM, E\$IIIPrm

**DM\_Exch - Exchange Data Between Drawmaps**

DM\_Exch exchanges the data in a rectangular shaped area between two drawmaps<sup>16</sup>. The two drawmaps must have the same data type. The source drawmap identifier is placed in the high-order word of register d3 and the destination drawmap identifier is placed in the low-order word. If the size would result in data being exchanged between the outsides of the two drawmaps, this size is reduced accordingly.

Input:           d0.w = Path number  
                  d1.w = SS\_DM setstat code  
                  d2.w = DM\_Exch function code  
                  d3.l = Drawmap identifiers (S:D)  
                  d4.l = Destination (H:V) coordinate  
                  d5.l = Source (H:V) coordinate  
                  d6.l = Size (W:H) in UCM coordinates

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BadMod, E\$UnID, E\$BPNuM, E\$IIIPrm

---

<sup>16</sup> If the areas to be exchanged have a part in common the final destination area is an exact copy of the initial source area and overlaps the final source area.

**DM\_TCpy - Copy with Transparency Check**

DM\_TCpy, like DM\_Copy, copies the data in a rectangular shaped area from one drawmap to another. As it copies it checks each source pixel value for equivalence to the specified transparent color. If the pixel value and the transparent color are equal, then the pixel is not copied. This call is usable only on CLUT and RGB555 drawmaps. The source drawmap identifier is placed in the high-order word of register d3 and the destination drawmap identifier is placed in the low-order word. If the specified size would result in data being copied from outside the source drawmap or to the outside of the destination drawmap, the size is reduced accordingly. It should be noted that the transparent color is interpreted according to the data type of the drawmap and is identical to the pixel data formats (see V.6.5).

Input:

- d0.w = Path number
- d1.w = SS\_DM setstat code
- d2.w = DM\_TCpy function code
- d3.l = Drawmap identifiers (S:D)
- d4.l = Destination (H:V) coordinate
- d5.l = Source (H:V) coordinate
- d6.l = Size (W:H) in UCM coordinates
- d7.w = Transparent color

Output: None

Error Output:

- cc = Carry bit set to one
- d1.w = Error code

Possible Errors: E\$BadMod, E\$UnID, E\$BPNuM, E\$IIIPrm

**DM\_TExc - Exchange with Transparency**

DM\_TExc, like DM\_Exch, exchanges the data in a rectangular shaped area between two drawmaps<sup>17</sup>. As it exchanges pixels, it checks each source pixel value for equivalence to the specified transparent color. If the pixel value and the transparent color are equal, the pixels are not exchanged. This function is usable only on CLUT and RGB555 drawmaps. The source drawmap identifier is placed in the high-order word of register d3 and the destination drawmap identifier is placed in the low-order word. If the specified size would result in data being exchanged between the outsides of the drawmaps, then the size is reduced accordingly.

Input:           d0.w = Path number  
                  d1.w = SS\_DM setstat code  
                  d2.w = DM\_TExc function code  
                  d3.l = Drawmap identifiers (S:D)  
                  d4.l = Destination (H:V) coordinate  
                  d5.l = Source (H:V) coordinate  
                  d6.l = Size (W:H) in UCM coordinates  
                  d7.w = Transparent color

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BadMod, E\$UnID, E\$BPNuM, E\$IIIPrm

---

<sup>17</sup> If the areas to be exchanged have a part in common, the final destination area is an exact copy of the initial source area and overlaps the final source area.

**DM\_Write - Write Drawmap**

DM\_Write writes a rectangular array of pixels, specified by the application and residing in its space, to a drawmap. Writing starts at the specified coordinate. The UCM assumes that the rows in the array, which correspond each to a line in a drawmap, are an integral number of long words in length. The number of pixels on each line are transferred as specified. However, when the UCM has finished writing a line of pixels, it skips to the next long word boundary and starts reading pixels there for the next line. The UCM also assumes that all of the bits for a pixel reside in a contiguous string of bits in the input array. If the application specifies a coordinate or size which will cause writing outside the drawmap an error (E\$IIIPrm) will be returned

Input:

- d0.w = Path number
- d1.w = SS\_DM setstat code
- d2.w = DM\_Write function code
- d3.w = Drawmap identifier
- d4.l = Start (H:V) coordinate
- d5.l = Size (W:H) of array<sup>18</sup> in UCM coordinates
- (a0) = Pointer to input pixel array

Output: None

Error Output:

- cc = Carry bit set to one
- d1.w = Error code

Possible Errors: E\$BadMod, E\$IIIPrm, E\$UnID, E\$BPNum

---

<sup>18</sup> The format of the input pixel array is resolution dependent: double resolution requires twice as much data as normal resolution

**DM\_IrWr - Irregular Write**

DM\_IrWr is used to write a series of lines of varying length to differing horizontal positions in the drawmap. Writing begins at the specified vertical position. The UCM assumes that each line in the input buffer is an integral number of long words in length. The specified number of pixels on each line are transferred; however, when UCM has finished writing a line of pixels, it skips to the next long word boundary and starts reading pixels there for the next line. The UCM also assumes that all of the bits for a pixel reside in a contiguous string of bits in the input array<sup>19</sup>.

If the application specifies a coordinate or size which will cause writing outside the drawmap the size is reduced accordingly.

Input:

- d0.w = Path number
- d1.w = SS\_DM setstat code
- d2.w = DM\_IrWr function code
- d3.w = Drawmap identifier
- d4.w = Starting vertical coordinate
- d5.w = Number of lines in UCM coordinates
- (a0) = Pointer to data to write
- (a3) = Pointer to transfer specifier array<sup>19</sup>

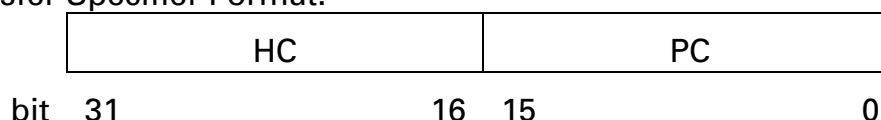
Output: None

Error Output:

- cc = Carry bit set to one
- d1.w = Error code

Possible Errors: E\$BadMod, E\$UnID, E\$BPNuM, E\$IIIPrm

Transfer Specifier Format:



- HC = Horizontal coordinate
- PC = Number of pixels to transfer in UCM coordinates

---

<sup>19</sup> The format of the array to transfer is resolution dependent: double resolution requires twice as much data as normal resolution.

**DM\_Read - Read Drawmap**

DM\_Read reads a rectangular area of pixels from a drawmap to a two dimensional array buffer in the application program's data space. Reading starts at the specified coordinate. As in the DM\_Write function, the UCM assumes that each line in the output array is an integral number of long words in length. Also DM\_Read packs all of the bits of a pixel into a contiguous string of bits in the output array. If the application specifies a coordinate or size which will cause reading outside the drawmap an error (E\$IIIPrm) will be returned.

Input:           d0.w = Path number  
                   d1.w = SS\_DM setstat code  
                   d2.w = DM\_Read function code  
                   d3.w = Drawmap identifier  
                   d4.l = Start (H:V) coordinate  
                   d5.l = Size (W:H) of area in UCM coordinates  
                   (a0) = Pointer to output area<sup>20</sup>

Output:           None

Error Output:    cc    = Carry bit set to one  
                   d1.w = Error code

Possible Errors: E\$BadMod, E\$IIIPrm, E\$UnID, E\$BPNum

---

<sup>20</sup> The format of the output area is resolution dependent: double resolution requires twice as much data as normal resolution.

**DM\_WrPix - Write Pixel**

DM\_WrPix writes the specified data to the pixel at the specified coordinate. If the application specifies a coordinate which would cause writing outside the drawmap, the data is not written and no error is returned. The pixel data written is in the lowest part of d5.w (e.g. for CLUT4).

Input:           d0.w = Path number  
                  d1.w = SS\_DM setstat code  
                  d2.w = DM\_WrPix function code  
                  d3.w = Drawmap identifier  
                  d4.l = Pixel (H:V) coordinate  
                  d5.w = Write data

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BadMod, E\$UnID, E\$BPNum



**DM\_RdPix - Read Pixel**

DM\_RdPix reads the pixel data at the specified coordinate. If the application specifies a coordinate which will cause reading outside the drawmap, an error (E\$IIIPrm) will be returned.

The pixel data read is put in the lower part of d0.w (e.g. for CLUT4).

Input:           d0.w = Path number  
                  d1.w = SS\_DM setstat code  
                  d2.w = DM\_RdPix function code  
                  d3.w = Drawmap identifier  
                  d4.l = Pixel (H:V) coordinate

Output:           d0.w = Pixel data

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BadMod, E\$IIIPrm, E\$UnID, E\$BPNum

**DM\_Cncl - Conceal Error in Drawmap**

DM\_Cncl will perform the error concealment function provided by the video driver. This will usually consist of replacement of an erroneous line of pixels by the previous line (this is actually done by copying the pointer from the previous entry in the line pointer table). If the first line is in error it is replaced by the first non-erroneous line in the drawmap. The error data input to the DM\_Cncl function is in the form of a bit string. There is one bit in this bit string for each line in the drawmap. If a particular bit is set to one then the line corresponding to that bit is in error.

Input:           d0.w = Path number  
                  d1.w = SS\_DM setstat code  
                  d2.w = DM\_Cncl function code  
                  d3.w = Drawmap identifier  
                  (a0) = Pointer to error data

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BadMod, E\$IIIPrm, E\$UnID, E\$BPNum

**DM\_Close - Close Drawmap**

DM\_Close de-allocates the memory associated with a drawmap.

Input:           d0.w = Path number  
                  d1.w = SS\_DM setstat code  
                  d2.w = DM\_Close function code  
                  d3.w = Drawmap identifier

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, F\$SRtMem, E\$BPNum

**DM\_DMDuP - Create Duplicate Drawmap Descriptor**

DM\_DMDuP creates a drawmap descriptor for the specified drawmap and copies the contents of the original drawmap descriptor into the new one. It also creates a new pattern and clipping region and copies the contents of the pattern and clipping region of the original drawmap descriptor into them. It does not create a new drawmap or line pointer table.

Input:           d0.w = Path number  
                  d1.w = SS\_DM setstat code  
                  d2.w = DM\_DMDuP function code  
                  d3.w = Drawmap identifier

Output:           d0.w = New drawmap identifier  
                  (a0) = Address of new drawmap descriptor

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IdFull, E\$UnID, F\$SRqCMem, F\$SRtMem, E\$BPNum

### 2.3.4.2 Graphics Cursor Functions

#### GC\_Pos - Position Graphics Cursor

GC\_Pos changes the position of the hardware graphics cursor on the display. The hit point coordinate specifies where the hit point of the graphics cursor will be positioned relative to the cursor origin.

Input:           d0.w = Path number  
                  d1.w = SS\_GC setstat code  
                  d2.w = GC\_Pos function code  
                  d3.l = Hit point (H:V) coordinate

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**GC\_Show - Show Graphics Cursor**

GC\_Show causes the hardware graphics cursor to appear on the screen.

Input:           d0.w = Path number  
                  d1.w = SS\_GC setstat code  
                  d2.w = GC\_Show function code

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**GC\_Hide - Hide Graphics Cursor**

GC\_Hide causes the hardware graphics cursor to disappear from the screen.

Input:           d0.w = Path number  
                  d1.w = SS\_GC setstat code  
                  d2.w = GC\_Hide function code

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**GC\_Ptn - Set Graphics Cursor Pattern**

GC\_Ptn controls the shape of the graphics cursor. The shape of the graphics cursor is specified using a bitmap, which can be of any size, limited by hardware capability. In the Base Case, the cursor is 16 x 16 pixels. Those pixels in the bitmap which are set to one will appear on the display in the cursor color (specified by the GC\_Col function). Those pixels that are set to zero will be transparent. The first word of the cursor pattern represents the first 16 pixels of the top line. Each line of pixels of the cursor is represented by an integral number of words in the cursor pattern input.

Input:

- d0.w = Path number
- d1.w = SS\_GC setstat code
- d2.w = GC\_Ptn function code
- d3.l = Hit point (H:V) position (relative to top left of cursor pattern)
- d4.l = Pattern (W:H) size in UCM coordinates
- d5.w = Cursor resolution
  - 0 = Normal
  - 1 = Double
  - 2 = High (extended case)
- (a0) = Pointer to graphics cursor pattern

Output: None

Error Output:

- cc = Carry bit set to one
- d1.w = Error code

Possible Errors: E\$BPNum



**GC\_Col - Set Graphics Cursor Color**

GC\_Col sets the color of the graphics cursor. The color is specified in terms of its Intensity, Red, Green, and Blue components at eight bits per component. If the resolution is less than eight bits then the most significant bits are used. In the Base Case, the resolution is only one bit per component, thus only the most significant bit is used.

Note that to obtain the Base Case set of colors, when the resolution is 8 bits, the intensity must have the values \$7F and \$FF only and the other components must have the values \$00 and \$FF only (see V.5.12).

However, other values will not be regarded as errors, this guarantees compatibility with future possible extensions (more bits per color component).

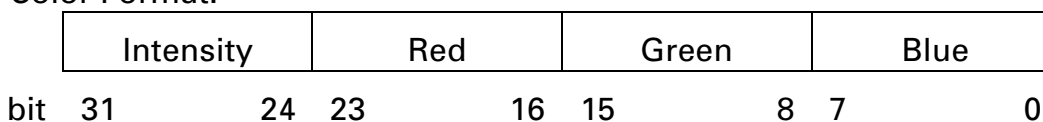
Input:           d0.w = Path number  
                   d1.w = SS\_GC setstat code  
                   d2.w = GC\_Col function code  
                   d3.l = Color

Output:           None

Error Output:   cc   = Carry bit set to one  
                   d1.w = Error code

Possible Errors: E\$BPNum

Color Format:



**GC\_Org - Set Graphics Cursor Origin**

GC\_Org sets the origin of the graphics cursor relative to the top left corner of the full screen image.

Input:           d0.w = Path number  
                  d1.w = SS\_GC Setstat code  
                  d2.w = GC\_Org function code  
                  d3.l = (H:V) origin (UCM) coordinates

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**GC\_Blnk - Set Graphics Cursor Blink Rate and Type**

The cursor is in its programmed color during the 'on' period, and the complementary color or transparent during the 'off' period. For 60Hz displays and 50Hz displays, the 'on' and 'off' periods are multiples of 200ms and 240ms respectively.

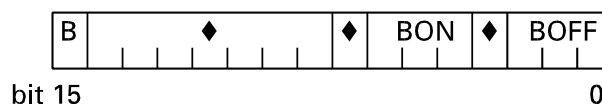
GC\_Blnk sets the blink rate of the graphics cursor.

Input:           d0.w = Path number  
                  d1.w = SS\_GC setstat code  
                  d2.w = GC\_Blnk function code  
                  d3.w = Blink constants

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**Blink Constants Format:**

- B                = Normal/Complement Blink  
0                = Normal  
1                = Complement
- BON             = Blink on period  
Period    = 12 \* BON fields \* frame time<sup>21</sup>  
BON        = 0 is not allowed
- BOFF            = Blink off period  
Period    = 12 \* BOFF fields \* frame time<sup>21</sup>  
If BOFF = 0, the cursor is on indefinitely

---

<sup>21</sup> Frame time = (1/display frequency)

## VII.2 File Managers

---

### 2.3.4.3 Regions

Regions divide a drawmap into two disjoint sets of pixels. Their primary use is to limit the area of drawing in a drawmap. Regions can also be drawn. Regions may be of any arbitrary shape or size. Regions are created from the basic shapes of rectangle, polygon, circle, circular wedge, ellipse, and elliptical wedge. More complex shapes are created from these basic shapes using intersection, union, exclusive-or, and difference operations. (see VII.2.3.2.6 for more information).

**RG\_Creat - Create Region**

RG\_Creat creates a region using one of the basic shapes.

Input:           d0.w = Path number  
                   d1.w = SS\_RG setstat code  
                   d2.w = RG\_Creat function code  
                   d3.w = Region type (rectangle, polygon, etc.)  
                   d4.l = Region type dependent  
                   d5.l = Region type dependent  
                   d6.l = Region type dependent  
                   d7.l = Region type dependent  
                   (a0) = Region type dependent

## Region Type Codes:

-1 = Null region: cannot be used in RG_Creat	
0 = Rectangle	6 = Elliptical wedge
1 = Elliptical Cornered Rectangle	7 = Predefined
2 = Polygon	8 = Border Fill (see DR_BFil)
3 = Circle	9 = Flood Fill (see DR_FFil)
4 = Circular wedge	10 = Complex result of a mix function: cannot be used in RG_Creat
5 = Ellipse	

## Parameters for Rectangular Region:

d4.l = Initial corner (H:V) coordinate  
 d5.l = Opposite corner (H:V) coordinate

## Parameters for Elliptical Cornered Rectangle:

d4.l = Initial corner (H:V) coordinate  
 d5.l = Opposite corner (H:V) coordinate  
 d6.l = Radii (H:V) of corner curvature

## Parameters for Polygon Region:

d4.w = Number of vertices in input array  
 (a0) = Pointer to input array

## Parameters for Circular Region:

d4.l = Center point (H:V) coordinate  
 d5.w = Radius

## Parameters for Circular Wedge Region:

d4.l = Start angle (H:V) coordinate  
 d5.l = End angle (H:V) coordinate  
 d6.l = Center point (H:V) coordinate  
 d7.w = Radius

VII.2 File Managers

---

## Parameters for Elliptical Region:

d4.l = Center point (H:V) coordinate  
 d5.l = Radii (H:V)

## Parameters for Elliptical Wedge Region:

d4.l = Start angle (H:V) coordinate  
 d5.l = End angle (H:V) coordinate  
 d6.l = Center point (H:V) coordinate  
 d7.l = Radii (H:V)

## Parameters for Predefined Region:

(a0) = Address of Region Data Structure

## Parameters for Border Fill Region:

d3.l = Drawmap Identifier : Region Type  
 d4.l = Interior (H:V) Coordinate  
 d5.w = Boundary Color

## Parameters for Flood Fill Region:

d3.l = Drawmap Identifier : Region Type  
 d4.l = Interior (H:V) Coordinate

## Output:

d0.w = Region identifier  
 (a0) = Address of region data structure

## Error Output:

cc = Carry bit set to one  
 d1.w = Error code

Possible Errors: E\$BadMod, E\$StkOvf, E\$IIIPrm, E\$IdFull, E\$RgFull,  
 F\$SRqMem, F\$SRtMem, E\$UnID, E\$BPNum

**RG\_Isect - Region Intersection**

RG\_Isect creates a new region which contains the set of points that are common to both of the input regions. If the two input regions have no common points then the new region will be empty.

Input:           d0.w = Path number  
                  d1.w = SS\_RG setstat code  
                  d2.w = RG\_Isect function code  
                  d3.w = First region identifier  
                  d4.w = Second region identifier

Output:           d0.w = New region identifier  
                  (a0) = Address of region data structure

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, F\$SRqMem, E\$RgFull, E\$IIdFull, E\$BPNum

**RG\_Union - Region Union**

RG\_Union creates a new region which combines the sets of points in both of the input regions.

Input:           d0.w = Path number  
                  d1.w = SS\_RG setstat code  
                  d2.w = RG\_Union function code  
                  d3.w = First region identifier  
                  d4.w = Second region identifier

Output:           d0.w = New region identifier  
                  (a0) = Address of region data structure

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, F\$SRqMem, E\$RgFull, E\$IIdFull, E\$BPNum



**RG\_Diff - Region Difference**

RG\_Diff creates a new region which contains the set of points in the first region with the set of points which are common to both regions removed. This implies that if the two input regions are equivalent then the output region will be empty.

Input:           d0.w = Path number  
                  d1.w = SS\_RG setstat code  
                  d2.w = RG\_Diff function code  
                  d3.w = First region identifier  
                  d4.w = Second region identifier

Output:           d0.w = New region identifier  
                  (a0) = Address of region data structure

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, F\$SRqMem, E\$RgFull, E\$ldFull, E\$BPNum

**RG\_XOR - Region Exclusive Or**

RG\_XOR creates a new region which contains the set of points in the two source regions that are not common to both of the input regions.

Input:           d0.w = Path number  
                  d1.w = SS\_RG setstat code  
                  d2.w = RG\_XOR function code  
                  d3.w = First region identifier  
                  d4.w = Second region identifier

Output:           d0.w = New region identifier  
                  (a0) = Address of region data structure

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, F\$SRqMem, E\$RgFull, E\$IIdFull, E\$BPNum

**RG\_Move - Move Region**

RG\_Move changes the position of the upper-left corner of the region's boundary rectangle to the specified coordinate. The boundary rectangle is a rectangle which completely encloses the region.

Input:           d0.w = Path number  
                  d1.w = SS\_RG setstat code  
                  d2.w = RG\_Move function code  
                  d3.w = Region identifier  
                  d4.l = New (H:V) coordinate

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$BPNuM

**RG\_Del - Delete Region**

RG\_Del releases the memory space associated with a region.

Input:           d0.w = Path number  
                  d1.w = SS\_RG setstat code  
                  d2.w = RG\_Del function code  
                  d3.w = Region identifier

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, F\$SRtMem, E\$BPNum

VII.2 File Managers

---

**2.3.4.4 Drawing Parameter Functions**

This set of functions change variables such as pattern and color which are used by the graphics drawing functions. All graphics display devices are required to support these functions. All parameters are set for each drawmap individually.

**DP\_Ptn - Set Drawing Pattern**

DP\_Ptn sets the drawing pattern which is used for most drawing operations. The pattern may contain actual pixel data or from one to four bits per pixel. In the latter mode, the pixel value is used as an index into a set of color registers which contain the actual data which will be written to the drawmap. For more information, refer to Pattern Indirect Drawing (VII.2.3.2.8).

Input:           d0.w = Path number  
                   d1.w = SS\_DP setstat code  
                   d2.w = DP\_Ptn function code  
                   d3.w = Drawmap identifier  
                   d4.w = Data type  
                   (a0) = Pointer to pattern

Output:           None

Error Output:   cc     = Carry bit set to one  
                   d1.w = Error code

Possible Errors: E\$BadMod, E\$UnID, F\$SRqCMem, F\$SRtMem, E\$BPNum

Data type Codes:

0 = Single-bit	4 = CLUT7
1 = Double-bit	5 = CLUT8
2 = Quad-bit	6-8 = Reserved
3 = CLUT4	9 = RGB555
	10-15 = Reserved

**DP\_PAIn - Set Pattern Alignment**

DP\_PAIn controls the correspondence between pattern pixels and drawmap pixels. The correspondence is made by specifying which pixel in the pattern corresponds to the pixel at the coordinate (0,0) in the drawmap. The input pattern alignment coordinates are taken modulo 16. For more information, refer to Pattern Indirect Drawing (VII.2.3.2.8).

Input:           d0.w = Path number  
                  d1.w = SS\_DP setstat code  
                  d2.w = DP\_PAIn function code  
                  d3.w = Drawmap identifier  
                  d4.l = Pattern alignment (H:V) position

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$BPNuM

**DP\_SCMM - Set Character Code Mapping Method**

DP\_SCMM sets the way that the DR\_Text and DR\_JTxt functions will interpret the character codes passed to them (e.g. how they will translate them to glyph numbers). The possible methods are the eight bit, seven/fifteen bit, and sixteen bit methods (0 = eight bit, 1 = seven/fifteen bit, and 2 = sixteen bit; see VII.2.3.2.9).

Input:           d0.1 = Path number  
                  d1.w = SS\_DP setstat code  
                  d2.w = DP\_SCMM function code  
                  d3.w = Drawmap identifier  
                  d4.w = Mapping method

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNum

**DP\_SCR - Set Color Register**

DP\_SCR sets color data in one color register to be used for pixel expansion (see VII.2.3.2.8). There are sixteen registers available numbered 0 to 15. The format of the color data is variable depending on the data type of the associated drawmap<sup>22</sup> and is identical to the pixel data formats (See V.6.5).

Input:           d0.w = Path number  
                  d1.w = SS\_DP setstat code  
                  d2.w = DP\_SCR function code  
                  d3.w = Drawmap identifier  
                  d4.w = Color register number  
                  d5.w = Color data

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BadMod, E\$BPNum

---

<sup>22</sup> The corresponding entry in the MD\_CRTbl of the drawmap descriptor is written 4 times with the color data for CLUT4 drawmaps, 2 times with the color data for CLUT7 and CLUT8 drawmaps.



**DP\_GFnt - Get Font**

DP\_GFnt makes a font module available for use by an application using the text functions. It will first try to link to the font module in memory. If this is unsuccessful, it will attempt to load it from the process' execution directory on the disc.

Input:           d0.w = Path number  
                  d1.w = SS\_DP setstat code  
                  d2.w = DP\_GFnt function code  
                  (a0) = Pointer to font module name

Output:           d0.w = Font identifier

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$MNF, F\$Link, F\$Load, F\$UnLink, F\$SRqMem, F\$SRtMem,  
                  E\$BPNum, E\$ldFull

The name of the standard CD-I font module, according to the ISO 8859-1 character set, is **font8x8**.

**DP\_AFnt - Activate Font**

DP\_AFnt sets the fonts that the DR\_Text and DR\_JTtxt functions will select glyphs from. Since these functions may select from up to four fonts, each font set by this call is associated with an active font number in the range of 0 to 3.

Input:           d0.w = Path number  
                  d1.w = SS\_DP setstat code  
                  d2.w = DP\_AFnt function mode  
                  d3.w = Drawmap identifier  
                  d4.w = Font identifier  
                  d5.w = Active font number

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BadMod, E\$BPNum

**DP\_DFnt - Deactivate Font**

DP\_DFnt deactivates a font activated by the DP\_AFnt function. The font which is associated with the specified active font number is deactivated.

Input:           d0.w = Path number  
                  d1.w = SS\_DP setstat code  
                  d2.w = DP\_DFnt function code  
                  d3.w = Drawmap identifier  
                  d4.w = Active font number

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$BPNuM, E\$IIIPrm

**DP\_RFnt - Release Font**

DP\_RFnt releases the memory space occupied by a font module.

Input:           d0.w = Path number  
                  d1.w = SS\_DP setstat code  
                  d2.w = DP\_RFnt function code  
                  d3.w = Font identifier

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, F\$UnLink, E\$BPNuM

**DP\_Clip - Set Clipping Region**

DP\_Clip sets the clipping region for subsequent graphics drawing. No drawing will be allowed outside of the region. Setting the region identifier to 0 causes the original boundary rectangle of the drawmap to be restored as the clipping region for the drawmap. If there is an existing clipping region, it will be deallocated after the new region is created.

Input:           d0.w = Path number  
                  d1.w = SS\_DP setstat code  
                  d2.w = DP\_Clip function code  
                  d3.w = Drawmap identifier  
                  d4.w = Region identifier

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$IIdFull, F\$SRqMem, F\$SRtMem, E\$BPNum

**DP\_PnSz - Set Pen Size**

This function is used to set the height and width of the lines that are drawn. This includes the 'lines' for unfilled rectangles, circles, arcs, etc.

Input:           d0.w = Path number  
                  d1.w = SS\_DP setstat code  
                  d2.w = DP\_PnSz function code  
                  d3.w = Drawmap identifier  
                  d4.l = Pen (W:H) size

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNum

**DP\_PStyl - Set Pen Style**

DP\_PStyl is used to define dash styling for lines. This includes, the "lines" for unfilled rectangles, circles, arcs, etc. The dash styling is represented by a bit string which determines the size of a dash and a number which represents the number of pixels each bit in the bit string represents.

Input:           d0.w = Path number  
                  d1.w = SS\_DP setstat code  
                  d2.w = DP\_PStyl function code  
                  d3.w = Drawmap identifier  
                  d4.l = Dash style bit string  
                  d5.w = Pixel count

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$BPNuM

**DP\_TCol - Set Transparent Color**

DP\_TCol sets the color value that will be used in the transparency check by the graphics drawing functions. It should be noted that the format of the color value is dependent on the data type of the drawmap<sup>23</sup> and is identical to the pixel data formats (see V.6.5).

Input:           d0.w = Path number  
                  d1.w = SS\_DP setstat code  
                  d2.w = DP\_TCol function code  
                  d3.w = Drawmap identifier  
                  d4.w = Transparent color

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$BadMod, E\$BPNuM

---

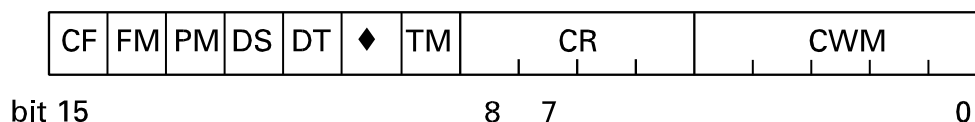
<sup>23</sup> The MD\_TColl entry of the drawmap descriptor is written 4 times with the color value data for CLUT 4 drawmaps, 2 times with the color value data for CLUT 7 and CLUT 8 drawmaps



## VII.2 File Managers

## 2.3.4.5 Graphics Drawing Functions

This set of functions execute graphics drawing such as line and circle. All of these functions have an operation code as a parameter, which causes them to behave in various ways. The operation code determines whether clipping will be performed, how shapes such as circles and polygons will be filled, and how pixels will be combined. Most functions in this section specify that a drawmap identifier and the operation code are placed in one register. The operation code is placed in the high-order word and the drawmap identifier is placed in the low-order word. The operation code is a sixteen bit word whose format is as follows (binary values).



- CF - Clipping Flag
  - 0 = Clipping off
  - 1 = Clipping on
- FM - Fill Mode
  - 0 = Outlined
  - 1 = Filled
- PM - Pattern Mode
  - 0 = Solid
  - 1 = Patterned
- DS - Dash Styling
  - 0 = Not dashed
  - 1 = Dashed
- DT - Dash transparency
  - 0 = off
  - 1 = on (transparent)
- TM - Transparency Writing Mode
  - 0 = No transparency
  - 1 = Transparency
- CR - Color register number for solid color

VII.2 File Managers

---

## CWM - Combinatorial Writing Modes

00000 = Write 0's  
00001 = Source AND Destination  
00010 = Source AND (NOT Destination)  
00011 = Replace  
00100 = (NOT Source) AND Destination  
00101 = Don't Modify  
00110 = Source XOR Destination  
00111 = Source OR Destination  
01000 = NOT (Source OR Destination)  
01001 = NOT (Source XOR Destination)  
01010 = NOT Destination  
01011 = Source OR (NOT Destination)  
01100 = NOT Source  
01101 = (NOT Source) OR Destination  
01110 = NOT (Source AND Destination)  
01111 = Write 1's  
10000 = Source + Destination  
10001 = Source - Destination  
10010 = Destination - Source  
10011 to 11111 are Reserved

Not all drawing functions use all these operation code fields. The bits of any unused fields must be set to zero.

**Note:** When using Combinatorial Writing modes except Replace, it is not guaranteed that a "closed figure" is actually closed.

**DR\_Dot - Draw a Dot**

DR\_Dot draws a dot using the current pen size and pattern at the specified coordinate.

Input:                   d0.w = Path number  
                          d1.w = SS\_DR setstat code  
                          d2.w = DR\_Dot function code  
                          d3.l = Drawmap identifier and Operation code  
                          d4.l = Drawing (H:V) coordinate

Output:                   None

Error Output:           cc    = Carry bit set to one  
                          d1.w = Error code

Possible Errors:   E\$UnID, E\$BPNuM

Operation code fields not used : FM

**DR\_Line - Draw a Line**

DR\_Line draws a line from the specified start coordinate to the specified end coordinate. The pixel at the end coordinate is not drawn.

Input:               d0.w = Path number  
                      d1.w = SS\_DR setstat code  
                      d2.w = DR\_Line function code  
                      d3.l = Drawmap identifier and Operation code  
                      d4.l = Start (H:V) coordinate  
                      d5.l = End (H:V) coordinate

Output:             None

Error Output:      cc    = Carry bit set to one  
                      d1.w = Error code

Possible Errors:   E\$UnID, E\$BPNUM

Operation code fields not used : FM

**DR\_PLin - Draw a Polyline**

DR\_PLin draws a series of lines starting at the first coordinate in the input array and ending at the last coordinate in the input array. The input array is an array of 16-bit integers. Each pair of integers specifies the horizontal and vertical coordinates of a vertex of the polyline. The pixel at the last specified point is not drawn.

Input:

- d0.w = Path number
- d1.w = SS\_DR setstat code
- d2.w = DR\_PLin function code
- d3.l = Drawmap identifier and Operation code
- d4.w = Number of vertices in input array
- (a0) = Input array pointer

Output: None

Error Output:

- cc = Carry bit set to one
- d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNum

Operation code fields not used : FM

**DR\_CArc - Draw a Circular Arc**

DR\_CArc draws a circular arc. The arc is specified by a center point, a radius, and two angle points. All of the points on the arc are equidistant (specified by the radius) from the specified center point. The video driver determines the starting point of the arc from a line which passes from the center point through the start angle point. It is the intersection of this line and a circle of the specified radius and the specified center point that is the calculated start point. The end point is calculated in exactly the same way from the line that starts at the center point and passes through the end angle point. Arcs are always drawn clockwise from the start to the end point.

Input:                   d0.w = Path number  
                          d1.w = SS\_DR setstat code  
                          d2.w = DR\_CArc function code  
                          d3.l = Drawmap identifier and Operation code  
                          d4.l = Start angle (H:V) coordinate  
                          d5.l = End angle (H:V) coordinate  
                          d6.l = Center point (H:V) coordinate  
                          d7.w = Radius in UCM coordinates

Output:                   None

Error Output:           cc    = Carry bit set to one  
                          d1.w = Error code

Possible Errors:   E\$IIPrm, E\$UnID, E\$BPNum

Operation code fields not used : FM

**DR\_EArc - Draw an Elliptical Arc**

DR\_EArc draws an elliptical arc. The arc is specified by a center point, two radii, and two angle points. All of the points on the arc lie on an ellipse with the specified center point and specified radii. The video driver determines the starting point of the arc from a line which passes from the center point through the start angle point. It is the intersection of this line and an ellipse of the specified radii and the specified center point that is the calculated start point. The end point is calculated in exactly the same way from the line that starts at the center point and passes through the end angle point. Arcs are always drawn clockwise from the start to the end point.

Input:

- d0.w = Path number
- d1.w = SS\_DR setstat code
- d2.w = DR\_EArc function code
- d3.l = Drawmap identifier and Operation code
- d4.l = Start angle (H:V) coordinate
- d5.l = End angle (H:V) coordinate
- d6.l = Center point (H:V) coordinate
- d7.l = Radii (H:V)

Output: None

Error Output:

- cc = Carry bit set to one
- d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNuM

Operation code fields not used : FM

**DR\_Rect - Draw a Rectangle**

DR\_Rect draws a rectangle, one corner of which is at the initial corner coordinate, the opposite corner at the opposite corner coordinate.

Input:                   d0.w = Path number  
                          d1.w = SS\_DR setstat code  
                          d2.w = DR\_Rect function code  
                          d3.l = Drawmap identifier and Operation code  
                          d4.l = Initial corner (H:V) coordinate  
                          d5.l = Opposite corner (H:V) coordinate

Output:                   None

Error Output:           cc    = Carry bit set to one  
                          d1.w = Error code

Possible Errors:       E\$UnID, E\$BPNuM



**DR\_ERect - Draw an Elliptical Corner Rectangle**

DR\_ERect draws a rectangle which has its corners replaced by ninety degree arcs of an ellipse with the specified radii. The rectangle is placed in the same way as a regular rectangle. The rectangle must be minimally twice as wide as the specified horizontal radius and twice as high as the vertical radius.

Input:

- d0.w = Path number
- d1.w = SS\_DR setstat code
- d2.w = DR\_ERect function code
- d3.l = Drawmap identifier and Operation code
- d4.1 = Initial corner (H:V) coordinate
- d5.1 = Opposite corner (H:V) coordinate
- d6.1 = Radii (H:V) of corner curvature

Output: None

Error Output:

- cc = Carry bit set to one
- d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNuM

**DR\_PGon - Draw a Polygon**

DR\_PGon draws a polygon. The lines of the polygon are calculated in the same way as the lines in DR\_PLin.

Input:                   d0.w = Path number  
                          d1.w = SS\_DR setstat code  
                          d2.w = DR\_PGon function code  
                          d3.l = Drawmap identifier and Operation code  
                          d4.w = Number of vertices in input array  
                          (a0) = Pointer to input array

Output:                 None

Error Output:       cc    = Carry bit set to one  
                      d1.w = Error code

Possible Errors:   E\$UnID, F\$SRqCMem, F\$SRtMem, E\$BPNum, E\$IIPrm

**DR\_Circ - Draw a Circle**

DR\_Circ draws a circle of the specified radius whose center point is at the specified coordinate.

Input:                   d0.w = Path number  
                          d1.w = SS\_DR setstat code  
                          d2.w = DR\_Circ function code  
                          d3.l = Drawmap identifier and Operation code  
                          d4.l = Center point (H:V) coordinate  
                          d5.w = Radius in UCM coordinates

Output:                   None

Error Output:           cc    = Carry bit set to one  
                          d1.w = Error code

Possible Errors:   E\$IIIPrm, E\$UnID, E\$BPNum

**DR\_CWdg - Draw a Circular Wedge**

DR\_CWdg draws a circular wedge. The arc part of the circular wedge is calculated in the same way as the circular arc in DR\_CArc.

Input:

- d0.w = Path number
- d1.w = SS\_DR setstat code
- d2.w = DR\_CWdg function code
- d3.l = Drawmap identifier and Operation code
- d4.l = Start angle (H:V) coordinate
- d5.l = End angle (H:V) coordinate
- d6.l = Center point (H:V) coordinate
- d7.w = Radius in UCM coordinates

Output: None

Error Output:

- cc = Carry bit set to one
- d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNum

**DR\_Elps - Draw an Ellipse**

DR\_Elps draws an ellipse whose center point is at the specified coordinate. The size and shape of the ellipse is specified by the ellipse's radii on the horizontal and vertical axes.

Input:               d0.w = Path number  
                      d1.w = SS\_DR setstat code  
                      d2.w = DR\_Elps function code  
                      d3.l = Drawmap identifier and Operation code  
                      d4.l = Center point (H:V) coordinate  
                      d5.l = Radii (H:V)

Output:               None

Error Output:       cc    = Carry bit set to one  
                      d1.w = Error code

Possible Errors:    E\$IIIPrm, E\$UnID, E\$BPNum

**DR\_EWdg - Draw an Elliptical Wedge**

DR\_EWdg draws an elliptical wedge. The arc part of the elliptical wedge is calculated in the same way as the elliptical arc in DR\_EArc.

Input:

- d0.w = Path number
- d1.w = SS\_DR setstat code
- d2.w = DR\_EWdg function code
- d3.l = Drawmap identifier and Operation code
- d4.l = Start angle (H:V) coordinate
- d5.l = End angle (H:V) coordinate
- d6.l = Center point (H:V) coordinate
- d7.l = Radii (H:V)

Output: None

Error Output:

- cc = Carry bit set to one
- d1.w = Error code

Possible Errors: E\$IIIPrm, ESUnID, E\$BPNum

**DR\_DRgn - Draw a Region**

DR\_DRgn draws a region which was created with the region functions.

Input:           d0.w = Path number  
                  d1.w = SS\_DR setstat code  
                  d2.w = DR\_DRgn function code  
                  d3.l = Drawmap identifier and Operation code  
                  d4.w = Region identifier

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$BPNuM

**DR\_BFil - Fill a Bounded Area with a Pattern**

DR\_BFil will fill an area which contains the specified coordinate and is bounded by all pixels which match the specified color.

Input:                d0.w = Path number  
                      d1.w = SS\_DR setstat code  
                      d2.w = DR\_BFil function code  
                      d3.l = Drawmap identifier and Operation code  
                      d4.l = Interior (H:V) coordinate  
                      d5.w = Boundary color

Output:                None

Error Output:        cc    = Carry bit set to one  
                      d1.w = Error code

Possible Errors:    E\$StkOvf, E\$IIPrm, F\$SRqCMem, F\$SRtMem, E\$BadMod,  
                      E\$BPNuM, E\$UnID

Operation code fields not used : FM, DS



**DR\_FFil - Flood Fill**

DR\_FFil changes the pixel at the specified coordinate and all pixels of the same color connected directly or indirectly to it using the pattern defined for the drawmap.

Input:           d0.w = Path number  
                  d1.w = SS\_DR setstat code  
                  d2.w = DR\_FFil function code  
                  d3.l = Drawmap identifier and Operation code  
                  d4.l = Fill start (H:V) coordinate

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$StkOvf, E\$IIIPrm, F\$SRqCMem, F\$SRtMem, E\$BPNum,  
                  E\$UnID

Operation code fields not used : FM, DS

**DR\_Copy - Copy Data from Drawmap to Drawmap**

DR\_Copy copies a rectangular area of pixels from one drawmap to another (or the same) drawmap.

Input:           d0.w = Path number  
                  d1.w = SS\_DR setstat code  
                  d2.w = DR\_Copy function code  
                  d3.l = Source/Destination drawmap identifiers  
                  d4.l = Destination (H:V) coordinate  
                  d5.l = Source (H:V) coordinate  
                  d6.l = Source area (W:H) size  
                  d7.w = Operation code

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BadMOd, ESUnID, E\$BPNuM

Operation code fields not used : FM, PM, DS, CR

**DR\_Text - Output Graphics Text**

DR\_Text generates text using the activated fonts. If two or more activated fonts include characters with the same glyph number, the font with the lowest font number will be used. The baseline of the cell of the first character is placed at the specified coordinate. Undrawable characters are ignored. DR\_Text continues to draw characters until either a NUL (\$00) is found in the text string or until the number of characters specified (in bytes) in d5.l has been written or until the edge of the drawmap is reached, whichever condition is satisfied first.

Input:           d0.w = Path number  
                  d1.w = SS\_DR setstat code  
                  d2.w = DR\_Text function code  
                  d3.l = Drawmap identifier and Operation code  
                  d4.l = Placement (H:V) coordinate  
                  d5.w = Number of character bytes to draw  
                  (a0) = Text pointer

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIIPrm, E\$UnID, E\$NoFont, E\$BPNum

Operation code fields not used : FM, PM, DS, CR

**DR\_JTxt - Output Justified Graphics Text**

DR\_JTxt is similar to DR\_Text except that an additional parameter is used to specify the length the text is to be justified to. DR\_JTxt expands the text by inserting extra space between characters. If there are too many characters DR\_Text displays only those characters which will fit. Control characters are ignored and the string is terminated by the same conditions as for DR\_Text.

Input:

- d0.w = Path number
- d1.w = SS\_DR setstat code
- d2.w = DR\_JTxt function code
- d3.l = Drawmap identifier and Operation code
- d4.l = Placement (H:V) coordinate
- d5.w = Justification length (UCM coordinates)
- d6.w = Number of character bytes to draw
- (a0) = Text pointer

Output: None

Error Output:

- cc = Carry bit set to one
- d1.w = Error code

Possible Errors: E\$IIIPrm, E\$UnID, E\$NoFont, E\$BPNum

Operation code fields not used : FM, PM, DS, CR



**DC\_RdFCT - Read Field Control Table**

DC\_RdFCT reads the specified number of instructions contained in the specified FCT into the applications buffer.

If d5.w = 0, the driver will return the error code (E\$IIPrm) and no instructions are read.

Input:

- d0.w = Path number
- d1.w = SS\_DC setstat code
- d2.w = DC\_RdFCT function code
- d3.w = FCT identifier
- d4.w = Starting instruction number
- d5.w = Number of instructions to read
- (a0) = Buffer pointer

Output: None

Error Output:

- cc = Carry bit set to one
- d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNum

**DC\_WrFCT - Write Field Control Table**

DC\_WrFCT writes the specified number of instructions from the applications buffer into the specified FCT.

Input:           d0.w = Path number  
                  d1.w = SS\_DC setstat code  
                  d2.w = DC\_WrFCT function code  
                  d3.w = FCT identifier  
                  d4.w = Starting instruction number  
                  d5.w = Number of instructions to write  
                  (a0) = Buffer pointer

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNum

**DC\_RdFI - Read Field Control Table Instruction**

DC\_RdFI reads a single instruction from the field control table.

Input:           d0.w = Path number  
                  d1.w = SS\_DC setstat code  
                  d2.w = DC\_RdFI function code  
                  d3.w = FCT identifier  
                  d4.w = Instruction number

Output:           d0.1 = Instruction

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNum



**DC\_WrFI - Write Field Control Table Instruction**

DC\_WrFI writes a single instruction to the field control table.

Input:           d0.w = Path number  
                  d1.w = SS\_DC setstat code  
                  d2.w = DC\_WrFI function code  
                  d3.w = FCT identifier  
                  d4.w = Instruction number  
                  d5.1 = Instruction to write

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNum

**DC\_DIFCT - Delete Field Control Table**

DC\_DIFCT deletes the specified FCT, freeing the memory space associated with it.

Input:           d0.w = Path number  
                  d1.w = SS\_DC setstat code  
                  d2.w = DC\_DIFCT function code  
                  d3.w = FCT identifier

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, F\$SRtMem, E\$BPNum

**DC\_CrLCT - Create Line Control Table**

DC\_CrLCT allocates a line control table (LCT) and fills it with No-Operation instructions. The LCT contains instructions which the display control hardware will execute during horizontal retrace periods. The size of the created LCT is dependent on its resolution.

If the resolution flag indicates normal resolution, the LCT will be created with only half as many lines as indicated. The even and odd lines will be interleaved and will be referenced starting with line 0. If the flag indicates high resolution, the LCT will be created with the given number of lines and the even and odd lines will be separated. The even numbered lines will be placed before the odd numbered lines.

Input:

- d0.w = Path number
- d1.w = SS\_DC setstat code
- d2.w = DC\_CrLCT function code
- d3.w = Plane number (0 or 1)
- d4.w = Number of lines (in UCM coordinates)
- d5.b = Resolution (0 = Normal/Double,  
1 = High)

Output: d0.w = LCT identifier

Error Output: cc = Carry bit set to one  
d1.w = Error code

Possible Errors: E\$IldFull, F\$SRqMem, F\$SRtMem, E\$BPNum, E\$IllPrm

**Note:** The largest value of the d4.w parameter is 1024. This is the same for either normal or high resolution. This means that on each (interlaced) video field, only 512 lines of LCT are available.

**DC\_RdLI - Read Line Control Table Instruction**

DC\_RdLI reads a single instruction from the LCT.

Input:           d0.w = Path number  
                  d1.w = SS\_DC setstat code  
                  d2.w = DC\_RdLI function code  
                  d3.w = LCT identifier  
                  d4.w = Line number (in UCM coordinates)  
                  d5.w = Column number

Output:           d0.l = Instruction

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNum

**DC\_WrLI - Write Line Control Table Instruction**

DC\_WrLI writes a single instruction to the LCT.

Input:           d0.w = Path number  
                  d1.w = SS\_DC setstat code  
                  d2.w = DC\_WrLI function code  
                  d3.w = LCT identifier  
                  d4.w = Line number (in UCM coordinates)  
                  d5.w = Column number  
                  d6.1 = Instruction

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNum

**DC\_DILCT - Delete Line Control Table**

DC\_DILCT deletes the specified LCT, freeing the memory space associated with it.

Input:           d0.w = Path number  
                  d1.w = SS\_DC setstat code  
                  d2.w = DC\_DILCT function code  
                  d3.w = LCT identifier

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, F\$SRtMem, E\$BPNum

**DC\_RdLCT - Read Line Control Table**

DC\_RdLCT reads the specified lines and columns of instructions from the specified line control table into the application's buffer.

If the LCT was created in normal resolution, the driver will in general fill up two lines in the application's buffer for each line in the LCT: the instructions put into the application's buffer for a line  $n$  will come from line  $(n/2)$  in the physical LCT, where  $n/2$  is always rounded down.

The application specifies the number of buffer lines to fill from data in the physical LCT.

If the LCT was created for high resolution, the driver will reconstruct a contiguous array of lines for the application's buffer.

Input:

- d0.w = Path number
- d1.w = SS\_DC setstat code
- d2.w = DC\_RdLCT function code
- d3.w = LCT identifier
- d4.w = Starting line number (in UCM coordinates)
- d5.w = Starting column number
- d6.w = Number of buffer lines to fill from the LCT
- d7.w = Number of columns to read
- (a0) = Buffer pointer

Output: d0.w = Number of instructions read

Error Output: cc = Carry bit set to one  
d1.w = Error code

Possible Errors: E\$IIIPrm, E\$UnID, E\$BPNum

**DC\_WrLCT - Write Line Control Table**

DC\_WrLCT writes a list of instructions from the application's buffer to the specified lines and columns of the specified LCT.

If the LCT was created in normal resolution, the driver will in general write two lines from the application's buffer to the same line of the LCT: the instructions in line  $n$  of the application's buffer will be written into line  $(n/2)$  of the physical LCT, where  $n/2$  is always rounded down. The lines are written into the physical LCT in increasing order.

The application specifies the number of buffer lines to write into the physical LCT.

If the LCT was created for high resolution, the driver will separate the instructions for the application to assure that the proper instructions are placed in the appropriate lines of the physical LCT.

Input:

- d0.w = Path number
- d1.w = SS\_DC setstat code
- d2.w = DC\_WrLCT function code
- d3.w = LCT identifier
- d4.w = Starting line number (in UCM coordinates)
- d5.w = Starting column number
- d6.w = Number of lines in the buffer to write into the physical LCT
- d7.w = Number of columns to write
- (a0) = Buffer pointer

Output: None

Error Output:

- cc = Carry bit set to one
- d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNuM



**DC\_PRdLCT - Read Physical Line Control Table**

DC\_PRdLCT reads the specified lines and columns of instructions from the specified Line Control Table into the application's buffer.

DC\_PRdLCT reads the physical LCT directly without regard to resolution or separation of lines. The application references an absolute line number (as opposed to UCM coordinates) and specifies the absolute number of lines to read.

Input:

- d0.w = Path number
- d1.w = SS\_DC setstat code
- d2.w = DC\_PRdLCT function code
- d3.w = LCT identifier
- d4.w = Starting line number (in absolute value)
- d5.w = Starting column number
- d6.w = Number of lines to read (in absolute value)
- d7.w = Number of columns to read
- (a0) = Buffer pointer

Output: d0.w = Number of instructions read

Error Output: cc = Carry bit set to one  
d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNum

**Note:** When using DC\_PRdLCT, the application may not access a line number greater than 512.

**DC\_PWrlCT - Write Physical Line Control Table**

DC\_PWrlCT writes a list of instructions from the application's buffer to the specified lines and columns of the specified Line Control Table.

DC\_PWrlCT writes to the physical LCT directly without regard to resolution or separation of lines. The application references an absolute line number (as opposed to UCM coordinates) and specifies the absolute number of lines to write.

Input:

- d0.w = Path number
- d1.w = SS\_DC setstat code
- d2.w = DC\_PWrlCT function code
- d3.w = LCT identifier
- d4.w = Starting line number (in absolute value)
- d5.w = Starting column number
- d6.w = Number of lines to write (in absolute value)
- d7.w = Number of columns to write
- (a0) = Buffer pointer

Output: None

Error Output:

- cc = Carry bit set to one
- d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNUM

**Note:** When using DC\_PWrlCT, the application may not access a line number greater than 512.

**DC\_NOP - Write No Operation Instructions to LCT**

DC\_NOP writes No Operation instructions to the specified locations in the LCT.

Input:

- d0.w = Path number
- d1.w = SS\_DC setstat code
- d2.w = DC\_NOP function code
- d3.w = LCT identifier
- d4.w = Starting line number (in UCM coordinates)
- d5.w = Starting column number
- d6.w = Number of lines to write (in UCM coordinates)
- d7.w = Number of columns to write

Output: None

Error Output:

- cc = Carry bit set to one
- d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNum

**DC\_SSig - Send Signal on Video Interrupt**

DC\_SSig sets up a signal to be sent to the calling process after a specified number of video interrupts. (see V.5.6). This is done only once, after which another DC\_SSig request must be issued.

Input:           d0.w = Path number  
                  d1.w = SS\_DC setstat code  
                  d2.w = DC\_SSig function code  
                  d3.w = Signal code  
                  d4.w = Number of interrupts to ignore  
                          (d4.w = 0 means next interrupt)

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**DC\_Relea - Release Signal on Video Interrupt**

DC\_Relea cancels any previous DC\_SSig function which is still active.

Input:           d0.w = Path Number  
                  d1.w = SS\_DC setstat code  
                  d2.w = DC\_Relea function code

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**DC\_SetCmp - Set Display Compatibility Mode**

DC\_SetCmp permits the display of different image formats which are different from the display format. The Resolution mode must be set to one if the horizontal dimensions are not the same.

The difference between the origin of the full screen image and the origin of the full screen display is returned.

Input:           d0.w = Path number  
                  d1.w = SS\_DC setstat code  
                  d2.w = DC\_SetCmp function code  
                  d3.w = Resolution mode  
                          0 = Same resolution  
                          1 = Different resolution

Output:           d0.l = Offset (H:V) coordinate in Normal Resolution convention

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**DC\_Intl - Set Display Interlace Mode**

DC\_Intl sets the display interlace mode.

Input:           d0.w = Path number  
                  d1.w = SS\_DC setstat code  
                  d2.w = DC\_Intl function code  
                  d3.w = Interlace mode  
                          0 = No interlace  
                          1 = Interlace

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$IIIPrm

**DC\_FLnk - Link LCT to FCT**

DC\_FLnk links the specified LCT to the specified FCT.

Input:           d0.w = Path number  
                  d1.w = SS\_DC setstat code  
                  d2.w = DC\_FLnk function code  
                  d3.w = FCT identifier  
                  d4.w = LCT identifier  
                  d5.w = LCT line number (in UCM coordinates)

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, E\$BPNum



**DC\_LLnk - Link LCT to LCT**

DC\_LLnk links the second specified LCT to the first one. The link causes the display control hardware to retrieve instructions from the second LCT. It should be noted that if the application program is using the maximum number of instructions in the specified line of the first LCT, the link will replace the last instruction contained in that line of the LCT.

Input:           d0.w = Path number  
                  d1.w = SS\_DC setstat code  
                  d2.w = DC\_LLnk function code  
                  d3.w = First LCT identifier  
                  d4.w = First LCT line number (in UCM coordinates)  
                  d5.w = Second LCT identifier  
                  d6.w = Second LCT line number (in UCM coordinates)

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIIPrm, E\$UnID, E\$BPNum

**DC\_Exec - Execute Display Control Program**

DC\_Exec causes the display control hardware to continuously execute the display control programs associated with the specified FCTs in planes A and B. If either FCT identifier is zero, the default FCT is executed in the specified plane. This will switch the display off in that plane.

Input:           d0.w = Path number  
                   d1.w = SS\_DC setstat code  
                   d2.w = DC\_Exec function code  
                   d3.w = FCT identifier (Plane A)  
                   d4.w = FCT identifier (Plane B)

Output:           None

Error Output:   cc    = Carry bit set to one  
                   d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnID, F\$RqCMem, F\$SRtMem, E\$BPNuM

**DC\_SetAR - Set the Display Aspect Ratio**

The DC\_SetAR function sets the aspect ratio of the display device. If the CD-I system does not support this feature, the function will return an E\$UnkSvc error.

Input:           d0.w = Path number  
                   d1.w = SS\_DC setstat code  
                   d2.w = DC\_SetAR function code  
                   d3.l = Aspect ratio parameter  
                           0 = "4:3"  
                           1 = "16:9"

Output:           None

Error Output:   cc    = Carry bit set to one  
                   d1.w = Error code

Possible Errors: E\$IIPrm, E\$UnkSvc, E\$BPNuM

### 2.3.4.7 Video Inquiry Functions

This is a general class of functions which are used to determine the characteristics of the graphics device. Device dependent factors such as display resolution, display size, or displayed text length can be determined or calculated using these functions.

#### VIQ\_TxtL - Calculate Length of Text

VIQ\_TxtL calculates the length of a character string in UCM coordinates if it were to be drawn using DR\_Text. It also calculates the number of characters in the passed parameter string which will fit in the specified length. It is useful for the text placement functions DR\_Text and DR\_JTxt. The number of characters used is the minimum of the number specified in d5.l and the number of characters which precede a NUL character.

If there is no active font, E\$NoFont will be returned.

Input:           d0.w = Path number  
                  d1.w = SS\_VIQ getstat code  
                  d2.w = VIQ\_TxtL function code  
                  d3.w = Drawmap identifier  
                  d4.w = Possible length (UCM coordinates)  
                  d5.w = Number of characters in text string  
                  (a0) = Text pointer

Output:           d0.w = Length of string (UCM coordinates)  
                  d1.w = Number of characters that will fit

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$NoFont, E\$BPNum

**VIQ\_CPos - Return Relative Character Positions**

VIQ\_CPos takes a string of characters and returns in a buffer the relative horizontal position (in UCM coordinates) of each character, starting from the first, from the placement coordinate if it were to be drawn using the DR\_Text function. Each entry in the returned list is a 16-bit signed value. VIQ\_CPos continues to process characters until either a null is found in the text string or until the number of characters specified in d4.l are processed, whichever condition is satisfied first.

If there is no active font, E\$NoFont will be returned.

Input:           d0.w = Path number  
                  d1.w = SS\_VIQ getstat code  
                  d2.w = VIQ\_CPos function code  
                  d3.w = Drawmap identifier  
                  d4.w = Number of text characters in string  
                  (a0) = Text pointer  
                  (a3) = Pointer to buffer for relative positions list

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$NoFont, E\$BPNum

**VIQ\_JCPs - Return Character Positions for Justified text**

VIQ\_JCPs takes a string of characters and returns in a buffer the relative horizontal position of each character, as defined for VIQ\_CPos, if it were to be drawn using the DR\_JTxt function.

If there is no active font, E\$NoFont will be returned.

Input:           d0.w = Path number  
                  d1.w = SS\_VIQ getstat code  
                  d2.w = VIQ\_JCPs function code  
                  d3.w = Drawmap identifier  
                  d4.w = Justification length (in UCM coordinates)  
                  d5.w = Number of text characters  
                  (a0) = Text pointer  
                  (a3) = Pointer to buffer for relative positions list

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$NoFont, E\$BPNuM

**VIQ\_FDta - Return Font Data**

VIQ\_FDta returns the font data section of the specified font (see VII.2.3.2.10).

Input:           d0.w = Path number  
                  d1.w = SS\_VIQ getstat code  
                  d2.w = VIQ\_FDta function code  
                  d3.w = Font identifier  
                  (a0) = Pointer to buffer for font data section

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$BPNuM

**VIQ\_GDta - Return Glyph Data**

VIQ\_GDta returns the glyph data table entry for the specified glyph in the specified font. (see VII.2.3.2.10).

Input:           d0.w = Path number  
                  d1.w = SS\_VIQ getstat code  
                  d2.w = VIQ\_GDta function code  
                  d3.w = Font identifier  
                  d4.w = Glyph number  
                  (a0) = Pointer to buffer for glyph data

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIIPrm, E\$UnID, E\$BPNum

**VIQ\_RGInfo - Return Pointer to Region Data Structure**

VIQ\_RGInfo returns a pointer to the Region Data Structure for information purposes only (see VII.2.3.2.6).

Input:           d0.w = Path number  
                  d1.w = SS\_VIQ getstat code  
                  d2.w = VIQ\_RGInfo function code  
                  d3.w = Region identifier

Output:           (a0) = Pointer to region data structure

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$BPNUM



**VIQ\_PntR - Test if Point in Region**

VIQ\_PntR tests whether the specified coordinate is contained in the specified region or not.

Input:           d0.w = Path number  
                  d1.w = SS\_VIQ getstat code  
                  d2.w = VIQ\_PntR function code  
                  d3.w = Region identifier  
                  d4.1 = Point (H:V) coordinate

Output:           d0.w = Non-zero if true, zero if false

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$BPNuM

**VIQ\_RLoc - Inquire Region Location**

VIQ\_RLoc returns the boundary rectangle for the specified region.

Input:           d0.w = Path number  
                  d1.w = SS\_VIQ getstat code  
                  d2.w = VIQ\_RLoc function code  
                  d3.w = Region identifier

Output:          d0.l = Coordinate (H:V) of upper-left corner  
                  d1.l = Coordinate (H:V) of lower-right corner

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$BPNuM

**VIQ\_DMInfo - Return Pointer to Drawmap Descriptor**

VIQ\_DMInfo returns a pointer to the drawmap descriptor for information purposes only (see VII.2.3.4.1).

Input:           d0.w = Path number  
                  d1.w = SS\_VIQ getstat code  
                  d2.w = VIQ\_DMInfo function code  
                  d3.w = Drawmap identifier

Output:           (a0) = Pointer to drawmap descriptor

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$BPNuM

## VII.2 File Managers

---

### 2.3.4.8 Character Output Functions

The UCM also provides the capability of treating a drawmap in much the same way as the display screen of a CRT terminal. The standard CD-RTOS system calls `I$Write` and `I$WriteLn` are used to write text in this manner. Since the UCM supports multiple images (multiple drawmaps) and multiple fonts, functions are described in this section to set the drawmap and font for character output for a particular path before `I$WriteLn` and `I$Write` can be used.

For any particular drawmap the active fonts must be monospaced and have the same character cell size. The number of characters per line and lines per screen are determined by the character cell size of the active fonts. If new fonts with a different character cell size are activated all active fonts must first be deactivated. UCM will then recalculate the number of characters per line and lines per drawmap.

#### **I\$WriteLn - Write Line**

`I$WriteLn` is a standard CD-RTOS service request. It writes a string of characters until either a carriage return (`$OD`) is found or the specified number of characters is written.

If a carriage return (code `$OD`) is found, UCM appends a cursor down (code `$OA`, also named line feed).

Input:            `d0.w` = Path number  
                  `d1.l` = Number of characters to write  
                  `(a0)` = Text string pointer

Output:           `d1.l` = Number of characters written

Error Output:    `cc`    = Carry bit set to one  
                  `d1.w` = Error code

Possible Errors: `E$NoFont`, `E$NoDM`, `E$BPNum`

**I\$Write - Write**

I\$Write is a standard CD-RTOS service request. It writes a string of the specified number of characters.

Input:           d0.w = Path number  
                  d1.l = Number of characters to write  
                  (a0) = Text string pointer

Output:           d1.l = Number of characters written

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$NoFont, E\$NoDM, E\$BPNum

**CO\_COD - Set Character Output Drawmap**

CO\_COD sets the drawmap that will be used for character output by I\$Write or I\$WriteLn for the specified path.

Input:           d0.w = Path number  
                  d1.w = SS\_CO setstat code  
                  d2.w = CO\_COD function code  
                  d3.w = Drawmap identifier

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$UnID, E\$BPNUM

**CO\_SCMM - Set Character Mode Mapping Method**

CO\_SCMM sets the way that the I\$Write and I\$WriteLn function will interpret the character codes passed to them (e.g. how they will translate them to glyph numbers). The possible methods are the eight bit, seven/fifteen bit, and sixteen bit methods (0 = eight bit, 1 = seven/fifteen bit, and 2 = sixteen bit; see VII.2.3.2.9).

Input:           d0.w = Path number  
                  d1.w = SS\_CO setstat code  
                  d2.w = CO\_SCMM function code  
                  d3.w = Mapping method

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIPrm, E\$BPNum

**CO\_AFnt - Activate Font**

CO\_AFnt sets the fonts that the I\$Write and I\$WriteLn functions will select glyphs from. Since these functions may select from up to four fonts, each font set by this call is associated with an active font number in the range of 0 to 3. The fonts activated with this call must be monospaced fonts and have that same character cell size. To change cell size, all currently activated fonts must first be deactivated using the CO\_DFnt function. Only if all active fonts are de-activated and then one new one is activated, CO\_AFnt will reposition the cursor to the upper left corner of the display.

Input:           d0.w = Path number  
                  d1.w = SS\_CO setstat code  
                  d2.w = CO\_AFnt function code  
                  d3.w = Font identifier  
                  d4.w = Active font number

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIIPrm, E\$NoFont, E\$UnID, E\$BPNuM



**CO\_DFnt - Deactivate Font**

CO\_DFnt deactivates a font activated by the CO\_AFnt function. The font which is associated with the specified active font number is deactivated.

Input:           d0.w = Path number  
                  d1.w = SS\_CO setstat code  
                  d2.w = CO\_DFnt function code  
                  d3.w = Active font number

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$IIPrm, E\$BPNum

#### 2.3.4.9 CRT Terminal Emulation Control Codes

This section describes character codes that are to be used for such functions as cursor addressing, backspace, and carriage return.

In this section a "space character" is defined as a rectangle with color value of color register 0 of the color register table in the drawmap descriptor (set by DP\_SCR).

All character positions (e.g. top and bottom line and first and last column) in this section are related to the complete character output drawmap, not to the visible screen.

##### **Cursor Up - Code \$0B**

This character causes the cursor to be moved up one line in the same column. If the cursor is on the top line then it is not moved.

##### **Cursor Down - Code \$0A**

This character causes the cursor to be moved down one line in the same column. If the cursor is positioned on the bottom line, this code will cause the screen or window to scroll up one line and the bottom line is cleared to space characters. This code is also known as a line feed.

##### **Cursor Right - Code \$0C**

This character causes the cursor to move one position to the right. If the cursor is positioned on the last column and the auto wrap mode is on, it moves to the first position of the next line. If the cursor is on the last column of the bottom line and the auto wrap mode is on, this code will cause the screen or window to scroll up one line and the cursor moves to the first position of the bottom line which is cleared to space characters.

##### **Cursor Left - Code \$08**

This character causes the cursor to move one position to the left. If the cursor is positioned on the first column, it moves to the last position of the previous line. If the cursor is on the first column of the top line, it does not move. This code is also known as backspace.

---

**VII.2 File Managers**

---

**Cursor Home - Code \$1E**

This character causes the cursor to be moved to the first column of the top line.

**Carriage Return - Code \$0D**

This character causes the cursor to move to the first position of the current line.

**Cursor Address - Code Sequence \$1B \$30 CC RR**

This sequence moves the cursor to a new position. CC and RR above represent the column and row numbers respectively, biased by \$1F. For example, CC and RR for column 1(left-most column), row 1 (top row) would be \$20 \$20.

**Delete Line - Code Sequence \$1B \$31**

This sequence causes the line on which the cursor is positioned to be replaced by the line below it. All lines following the deleted line are scrolled up by one line. The bottom line is cleared to space characters.

**Insert Line - Code Sequence \$1B \$32**

This sequence causes the line on which the cursor is positioned and all lines following it to be scrolled down by one line. The line on which the cursor is positioned is cleared to space characters.

**Show Cursor - Code Sequence \$1B \$33**

This sequence causes the cursor to appear on the screen.

**Hide Cursor - Code Sequence \$1B \$34**

This sequence causes the cursor to disappear from the screen.

**Clear to End of Line - Code Sequence \$1B \$35**

This sequence causes all characters from the cursor position to the end of the line to be cleared to space characters.

**Clear to End of Screen - Code Sequence \$1B \$36**

This sequence causes all characters from the cursor position to the end of the drawmap to be cleared to space characters.

**Start Reverse Video - Code Sequence \$1B \$37**

All characters written following this sequence will appear on the display with each pixel in the glyph complemented before it is used to select color data from a color register or written to the drawmap.

**End Reverse Video - Code Sequence \$1B \$38**

All characters following this sequence will appear normally on the display. This sequence reverses the previous control sequence.

**Start Underlining - Code Sequence \$1B \$39**

All characters written following this sequence will appear on the display with an underline.

**End Underlining - Code Sequence \$1B \$3A**

All characters written following this sequence will appear on the display without an underline.

**Clear Screen - Code Sequence \$1B \$3B**

This sequence causes all character positions to be cleared to space characters. It does not cause the cursor to move.

**Insert Character - Code Sequence \$1B \$3C**

This sequence causes the character under the cursor and all characters to the right of it on the line to be moved to the right one character position. The character under the cursor is cleared to a space character. If there is a character in the last column, it is removed.

**Delete Character - Code Sequence \$1B \$3D**

This sequence causes all characters to the right of the cursor on the line to be moved to the left one character position. The character under the cursor is destroyed and the last character position on the line is cleared to a space character.

**Turn On Auto Wrap - Code Sequence \$1B \$3E**

After this sequence is given, when any character is written to the last character position of a line, the carriage return and cursor down functions will automatically be performed.

**Turn Off Auto Wrap - Code Sequence \$1B \$3F**

After this sequence is given, when any character is written to the last character position of a line, the cursor will not be moved as it would be if characters were written to previous character positions.

### 2.3.5 Pointer Input Functions

This set of functions is provided to access pointer type devices such as mice and graphics tablets. Several pointers may be accessed by an application simply by opening a path to the device associated with the pointer.

#### PT\_Coord - Obtain Pointer Coordinates

PT\_Coord returns the current coordinate of the pointer (with respect to the origin), the pointer state and the button state. The pointer active bit indicates, for graphics tablets, touch screens etc that the pointer is active. For pointers which are always active this bit will always be set to one.

Input:           d0.w = Path number  
                   d1.w = SS\_PT getstat code  
                   d2.w = PT\_Coord function code

Output:           d0.w = Pointer and Button state  
                   Meaning (when set to one):  
                   Bit 0 = 1: Button 1 depressed  
                   Bit 1 = 1: Button 2 depressed  
                   Bits 2 to 14 reserved  
                   Bit 15 = 1: Pointer active  
                   d1.l = Pointer (H:V) coordinate

Error Output:   cc    = Carry bit set to one  
                   d1.w = Error code

Possible Errors: E\$BPNum

**PT\_SSig - Send Signal on Pointer Change**

PT\_SSig sets up a signal to be sent to the calling process when the pointer is moved or when one of its triggers changes state.

Input:           d0.w = Path number  
                  d1.w = SS\_PT setstat code  
                  d2.w = PT\_SSig function code  
                  d3.w = User defined signal code

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$PBNuM, E\$NotRdy

**PT\_Relea - Release Device**

PT\_Relea releases the pointer from a PT\_SSig request that was made.

Input:           d0.w = Path number  
                  d1.w = SS\_PT setstat code  
                  d2.w = PT\_Relea function code

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum



**PT\_Org - Set Pointer Origin**

PT\_Org adjusts the origin of the pointing device within its coordinate space. The new origin is specified as an offset from the default origin which is the upper left corner of the full screen display.

Input:           d0.w = Path number  
                  d1.w = SS\_PT setstat code  
                  d2.w = PT\_Org function code  
                  d3.l = New origin (H:V) (High Resolution coordinates)

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**PT\_Pos - Set Pointer Position**

PT\_Pos adjusts the position of the pointer of relative pointing devices with respect to the origin of the pointing device. This call adjusts the coordinates returned by the pointer.

Input:           d0.w = Path number  
                  d1.w = SS\_PT setstat code  
                  d2.w = PT\_Pos function code  
                  d3.l = New position (H:V) (High Resolution coordinates)

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

### 2.3.6 Keyboard Input Functions

Data from the keyboard is retrieved using CD-RTOS's standard input function `I$Read`. Additional functions are provided to get more information and inquire about and control the status of the keyboard.

#### **I\$Read - Read String**

`I$Read` reads a string of characters from the keyboard device until the specified number of characters has been read. Note that UCM does not write these characters to the display.

Input:            `d0.w` = Path number  
                  `d1.l` = Number of characters to read  
                  `(a0)` = Buffer pointer

Output:           `d1.l` = Number of characters read

Error Output:    `cc`    = Carry bit set to one  
                  `d1.w` = Error code

Possible Errors: `E$BPNum`, `E$BMode`, `E$Read`, `E$EOF`, `E$NotRdy`

**KB\_Rdy - Check for Data Ready**

KB\_Ready returns the number of characters in the input buffer. These characters may relate to either key down or auto-repeat events; key-up events are ignored as far as this function is concerned.

Input:           d0.w = Path number  
                  d1.w = SS\_KB getstat code  
                  d2.w = KB\_Rdy function code

Output:           d1.l = Number of characters available

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$NotRdy (if no data available)

This function can be used to read all available keyboard data at one go, by using the returned value as the "count" argument to I\$Read.

**KB\_Read - Read Keyboard Event**

KB\_Read allows access to extended information about keyboard events. Events are queued and returned individually for key down, key up and auto repeat.

Input:           d0.w = Path number  
                   d1.w = SS\_KB getstat code  
                   d2.w = KB\_Read function code  
                   d3.w = mode  
                           bit 0 = sleep mode if set  
                           bit 1 = unstack mode if set

Output:           d0.w = character code  
                   d1.b = data type  
                           Meaning (when set to one):  
                           Bit 0 : key down  
                           Bit 1 : auto-repeat  
                           Bit 2 : key up

Error output:    cc    = Carry bit set to one  
                   d1.w = Error code

Possible Errors: E\$BPNum, E\$Read, E\$NotRdy

If the buffer is empty when this function is called, the action taken depends on the 'sleep mode' bit in d1; if it is zero, the function will return immediately with d1 = 0; if it is one, the caller will be put to sleep until an event arrives.

The unstack bit indicates whether the event is to be removed from the input buffer; hence, if this bit is not set, a subsequent identical call to KB\_Read will return the same event again.

**Important Notes:**

Once KB\_Read has been called, the driver will assume that the application intends to read all subsequent keyboard data using KB\_Read, not I\$Read. This affects the working of the function KB\_SSig as described below.

An application wishing to process all events currently in the input buffer may use repeated KB\_Read calls until a zero is returned in d1; KB\_Ready should not be used to 'count' the buffer since this ignores key-up events.

**KB\_SSig - Send Signal on Keyboard data ready**

KB\_SSig sets up a signal to be sent to the calling process when data from the keyboard is available. Normally, only a key-down or auto-repeat event will result in a signal; if, however, KB\_Read has previously been called on the path, a key-up event will also cause a signal.

Input:           d0.w = Path number  
                  d1.w = SS\_KB setstat code  
                  d2.w = KB\_SSig  
                  d3.w = User defined signal code

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$NotRdy

**Note:** A signal is also sent if keyboard data is available in the input buffer at the moment of the KB\_SSig request.

**KB\_Rel - Release device**

KB\_Rel releases the keyboard from a KB\_SSig request that was made.

Input:           d0.w = Path number  
                  d1.w = SS\_KB setstat code  
                  d2.w = KB\_Rel function code

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**KB\_Repeat - Set Keyboard Latency and Repeat Times**

KB\_Repeat allows an application to alter the keyboard auto repeat latency and repeat times to suit the preferences of a user. The times are specified in units of one system tick (10 milliseconds) unless the high order bit is set, in which case the rest of the long word specifies how many 256ths of a second.

**Note:** All times are rounded off to the nearest 0.01 second.

Input:           d0.w = Path number  
                  d1.w = SS\_KB setstat code  
                  d2.w = KB\_Repeat function code  
                  d3.l = Latency time  
                  d4.l = Repeat time

Output:           None

Error output:    cc    = carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

If the latency time is specified as zero, no auto repeat will be generated.

If the latency and repeat times are respectively non-zero and zero, only one auto repeat will be generated.



**KB\_Stat - Determine Status of Keyboard**

KB\_Stat returns information about the current state of the keyboard, including the state of the special keys.

Input:           d0.w = Path number  
                  d1.w = SS\_KB getstat code  
                  d2.w = KB\_Stat function code

Output:          d0.w = character code  
                  d1.b = special keys state  
                          Bit 0 : shift depressed when set  
                          Bit 1 : caps depressed when set  
                          Bit 2 : supershift depressed when set  
                          Bit 3 : control depressed when set

If the character code is 0, then no keys are being pressed.

If more than one key is depressed, d0.w will contain the character code of the last key pressed.

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

**Warning:**    This function relates to the actual keyboard state and doesn't affect the input buffer in any way. Hence, even if an application wishes to use exclusively KB\_Stat for keyboard polling, it should flush the buffer occasionally (e.g. by using KB\_Read in "unstack but don't sleep mode" until no event is returned) to avoid input over-run.

### 2.3.7 Player Control Key functions

The implementation of the Player Control Key functions is optional in the CD-I system. If only the minimum set of Player Control Keys is implemented (see A VII.2.5.3), then the `KB_Avail` and the `KB_NrAvail` calls are optional in the CD-I playback system. If more than the minimum set of Player Control Keys are implemented, then the `KB_Avail` and `KB_NrAvail` calls are mandatory.

Data from the Player Control Keys can be retrieved by using the CD-RTOS standard input functions `I$Read` and `KB_Read`. Additional functions are provided to get more information and inquire about and control the status of the keyboard.

#### **KB\_Avail - Key(s) available for the application**

`KB_Avail` will return the keycode(s) in keygroup 8 that are available to the application. The keycodes used are those as defined in the keygroup 8 code assignment.

The request can be done for a specific keycode. If the keycode returned in the buffer is the same as the requested then the keycode is implemented in the CD-I playback system. If not the call will return the next key available from a sorted list of keys. Every time this call is used with a non existing keycode in the keycode requested field, the next available keycode will be returned to the application. If there is not a next available key in the sorted list of keys then the call will return the first available key in the sorted list of keys.

The request can be done for a number of keycodes. The call will return the keycodes available starting with the keycode in the keycode requested field.

Input:           d0.w = Path number  
                   d1.w = SS\_KB getstat code  
                   d2.w = KB\_Avail function code  
                   d3.w = first keycode requested  
                   d4.w = number of keycodes to read  
                   (a0) = pointer to keycode buffer

Output:           d1.w = Number of keycodes actually read

Error Output:    cc    = Carry bit set to one  
                   d1.w = Error code

Possible Errors: `E$BPNum`, `E$UnkSvc`

**KB\_NrAvail - Number of keycodes available for the application**

KB\_Avail will return the number of keycodes in keygroup 8 that are available to the application. The keycodes used are those defined in the keygroup 8 code assignment.

Input:           d0.w = Path number  
                  d1.w = SS\_KB getstat code  
                  d2.w = KB\_NrAvail function code

Output:           d1.w = Number of keycodes available

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$UnkSvc

## VII.2 File Managers

---

### 2.4 Non-volatile RAM File Manager (NRF)

#### 2.4.1 General

The non-volatile RAM file manager (NRF) provides file level support for battery backed up system memory. NRF offers an alternative to RBF's 'RAM disks' when the amount of non-volatile RAM is small.<sup>24</sup> NRF provides less functionality than RBF and slower throughput than pipes, however, it consumes a much smaller amount of memory for file structure overhead.

Non-volatile RAM is often a very precious system resource. NRF is specifically customized to use as little of this memory as possible. Applications needing to keep small files in memory should always use RAM disk files or named pipes in preference to NRF files when possible.

---

<sup>24</sup> 'RBF' is the optional CD-RTOS random block file manager used with magnetic disks and random access memory files. RBF is described in detail in A VII.1.

VII.2 File Managers

---

**2.4.2 I\$Create - Creating NRF Files**

The I\$Create service request is used with the NRF manager to create new named files.

NRF files may be created by using the pathlist "/NVR/<name>" where <name> is the logical file name. If an NRF file with the same name already exists, an error (E\$CEF) is returned. Access permissions may be specified, and are handled similar to RBF<sup>3</sup>.

The application must specify an initial file size for NRF files when creating a new file. This is done by passing the desired file size in register d2. This technique allows NRF to ensure that enough space exists for the file, and speeds up subsequent write requests.

Input:           d0.w = Access mode (W R)  
                  d1.w = File attributes (PW PR W R)  
                  d2.l = Initial file size  
                  (a0) = Pathname pointer

Output:           d0.w = Path number  
                  (a0) = Updated past the pathlist

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$PthFul, E\$BPNam, E\$MemFul, E\$CEF

Access mode format:

Bit Number	Description (if set to one)
0	Read (R)
1	Write (W)
2-15	Reserved

File attributes format:

Bit Number	Description (if set to one)
0	Owner read (R)
1	Owner write (W)
2	Reserved
3	Public read (PR)
4	Public write (PW)
5-15	Reserved

VII.2 File Managers

---

**2.4.3 I\$Open - Opening NRF Files**

An existing NRF file may be accessed using the I\$Open service request, passing the same name specified when the file was created. I\$Open searches through a list of file names associated with the given NRF device descriptor. If the file name is not found, an error (E\$PNNF) is returned.

NRF enforces record locking for files on the NVRAM device. If an open request is made for an NRF file while it is being created, or is already open for write access, an error (E\$Busy) is returned. Similarly if a file is open for read access and a process attempts to open it for write access an error (E\$Busy) is returned. Several processes may have the same NRF file opened simultaneously only if they do not have write access to the file.

Input:           d0.w = Access mode  
                  (a0) = Pathname pointer

Output:           d0.w = Path number  
                  (a0) = Updated past pathname

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$PthFul, E\$BPNam, E\$BMode, E\$FNA, E\$PNNF, E\$Busy, E\$Share

Access mode format:

Bit Number	Description (if set to one)
0	Read (R)
1	Write (W)
2-5	Reserved
6	Non-shareable use
7	Directory

VII.2 File Managers

---

**2.4.4 I\$Delete - Deleting Files**

The I\$Delete service request is used to remove NRF files and return their memory to the system. A process must have write permission for any NRF file being deleted. If the file is not found or is currently open, an error (E\$PNNF or E\$Busy) is returned.

When an NRF file is deleted, other non-volatile RAM files will be compacted together to make the available memory contiguous.

Input:           d0.w = Access mode  
                 (a0) = Pathname pointer

Output:           (a0) = Updated past pathname

Error Output:    cc    = Carry bit set to one  
                 d1.w = Error code

Possible Errors: E\$BPNam, E\$PNNF, E\$BMode

Access Mode Format: See VII.2.4.2.

VII.2 File Managers

---

**2.4.5 I\$Seek - Random Access**

The I\$Seek service request may be used to access data within an NRF file randomly. No error is produced at the time of the seek if the new position is beyond the current 'end of file'.

Input:           d0.w = Path number  
                  d1.l = New file position

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum



VII.2 File Managers

---

**2.4.6 I\$Read, I\$ReadLn - Reading Data**

NRF supports I\$Read and I\$ReadLn for returning the specified number of data bytes from the current file position. I\$ReadLn terminates when a carriage return or the specified number of characters have been read, whichever comes first. An end of file error (E\$EOF) is returned when there is no more data in the file.

Input:           d0.w = Path number  
                  d1.l = Maximum number of bytes to read  
                  (a0) = Buffer pointer

Output:           d1.l = Number of bytes actually read

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$Read, E\$BMode, E\$EOF

### 2.4.7 **I\$Write, I\$WriteLn - Writing Data**

The I\$Write and I\$WriteLn service requests may be used to write data into an NRF file. I\$WriteLn terminates when a carriage return or the specified number of bytes have been read, whichever comes first. I\$Write requests beyond the current 'end of file' automatically expand the file with the new data. An error (E\$Write) is returned if there is not enough memory to satisfy the write request.

The data within an NRF file occupies one contiguous block of memory. Expanding a file may cause a reorganization of files within the non-volatile RAM.

Input:           d0.w = Path number  
                  d1.l = Maximum number of bytes to write  
                  (a0) = Buffer pointer

Output:           d1.l = Actual number of bytes written

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum, E\$BMode, E\$Write, E\$MemFul

### 2.4.8 **I\$Close - Closing NRF Files**

The I\$Close service request must be used to close an NRF file when it is no longer needed. Files open for write access must be closed before they may be accessed by any other process. An NRF file may not be deleted if it is open.

Input:               d0.w = Path number

Output:             None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$BPNum

VII.2 File Managers

---

**2.4.9 Status Functions**

In the I/O System Calls chapter of Appendix VII.1 a specification of the parameters and function of the status functions is given. Figure VII.21 below lists the status functions that are supported by NRF.

Figure VII.29 **Status functions supported by NRF****GetStat Functions:**

Name	Description
SS_Opt	Reads the 128-byte path descriptor option section
SS_Size	Returns the current file size
SS_EOF	Tests for end of file
SS_FD	Returns the file descriptor
SS_Pos	Get current file position
SS_FDInf	Get specified FD information
SS_Free	Return size of free space on device

**SetStat Functions:**

Name	Description
SS_Opt	Rewrites the 128-byte path descriptor option section
SS_Size	Expands or contracts the file to the specified size
SS_Attr	Changes the NRF file attributes

## VII.2 File Managers

---

### 2.4.10 Changing and Making Directories

The NRF file manager does not support directory files and returns an error (E\$UnkSvc) for both the I\$MakDir and I\$ChgDir service requests.

### 2.4.11 NRF Directories

It is possible to determine the contents of an NRF device by opening the NVRAM device in Directory mode. Subsequent read requests return a 32-byte record for each NRF file, in a format compatible with RBF and PIPE devices. This allows utilities that normally read an RBF directory file sequentially to work with NRF as well.

**2.5 Error Codes**

<b>Error Number</b>	<b>Description</b>
006:000 E\$IIPrm	Illegal Parameter
006:001 E\$IIdFull	Identifier (ID) table full
006:002 E\$BadSiz	Bad size error
006:003 E\$RgFull	Region definition full (overflow)
006:004 E\$UnID	Unallocated identifier number
006:005 E\$NullRg	Null Region
006:006 E\$BadMod	Bad drawmap/region/pattern mode
006:007 E\$NoFont	No active font
006:008 E\$NoDM	No drawmap
006:009 E\$NoPlay	No audio play in progress
006:010 E\$Abort	Play aborted

### VII.3 Device Drivers

---

#### VII.3 Device Drivers

##### 3.1 Compact Disc

###### 3.1.1 Introduction

This section specifies the interface between the Compact Disc File Manager (CDFM) and its drivers. As with all CD-RTOS device drivers, the CDFM drivers will perform basic low level physical input/output functions. The driver is not concerned with files, directories, etc., which are handled at a higher level by the CDFM. Device driver modules are re-entrant so one copy of the module can simultaneously support multiple devices that use identical I/O controller hardware.

###### 3.1.2 Basic Functional Requirements of CDFM Drivers

The CDFM device driver module is a package of six subroutines that are called by the CDFM in system state. The functions are:

- (1) Initialize the device controller hardware and related driver variables as required.
- (2) Read standard physical units.
- (3) Return a specified device status.
- (4) Set a specified device status.
- (5) De-initialize the device. It is assumed that the device will not be used again unless re-initialized.
- (6) Play a selected portion of the disc.

When written properly, a single physical driver module can handle multiple identical hardware interfaces. The specific information for each physical interface (port address, initialization constants, etc.) is given in a **device descriptor** module. Device descriptor modules are described in section VII.3.1.4.1.

The name by which the device is known to the system is the name of the device descriptor module. CD-RTOS copies the information contained in the device descriptor module to the **path descriptor** data structure and the device driver static storage area for easy access by the drivers.

### 3.1.3 Driver Module Format

All drivers must conform to the standard CD-RTOS memory module format. The module type code is "Drivr".

The execution offset in the module header (M\$Exec) gives the address of an **offset table**, which specifies the starting address of each of the seven driver subroutines. The driver subroutines are called by the CDFM through the offset table. The driver routines are always executed in system state. Each subroutine is terminated by an RTS instruction. Error status is returned using the CC carry bit with an error code return in register d1.w.

The **static storage size** (M\$Mem) specifies the amount of local storage required by the driver. This is the sum of the storage required by the file manager plus any variables and tables declared in the driver.



### VII.3 Device Drivers

---

#### 3.1.4 Data Structures

CDFM and its drivers must concern themselves with the following data structures:

- Device Descriptors
- Path Descriptors
- Device Static Storage
- Device Drive Tables
- PLAY Control Blocks
- PLAY Control Lists

##### 3.1.4.1 CDFM Device Descriptors

Device descriptor modules are small, non-executable modules that provide information that associates a specific I/O device with its logical name, hardware controller address(es), device driver name, file manager name, and initialization parameters.

Device drivers and file managers both operate on general classes of devices, not specific I/O ports. The device descriptor modules tailor their function to a specific I/O device. One device descriptor module must exist for each I/O device in the system. However, one device may also have several device descriptors with different initialization constants.

The name of the module is used as the logical device name by the system and the user. Its format consists of a standard module header (48 bytes) that has a module type code "Devic". The remaining header fields (24 bytes) are standard for device descriptor modules.

VII.3 Device Drivers

---

**3.1.4.1.1 Standard Device Descriptor Header Fields**

The first 72 bytes of each device descriptor are defined in the same way for all device descriptors in CD-RTOS. For further information refer to Appendix VII.1.

**3.1.4.1.2 Device Descriptor Option Fields (Initialization Table)**

The initialization table is copied into the 'option section' of the path descriptor when a path to the device is opened. It is generally referenced using the path descriptor. Additional option fields may be added here. The values in this table may be used to define the operating parameters that are accessible by the I\$GetStt and I\$SetStt system calls of SS\_Opt. The maximum size of the initialization table is 128 bytes.

Figure VII.30 **Device Descriptor Option Fields**

Offset	Length	Name	Description
72	1	PD_DTP	Device class (device class of CDFM is 5).
73	1	PD_CDFC	Functional Class (0=CD, 1=Audio)
74	1	PD_ErrSz	Number of bytes per error bit
75	1	PD_NDscs	Number of discs in player
76	1	PD_DNum	Device number for CD Drive or Audio Processor

**3.1.4.1.3 Device Configuration Fields**

There is a section of the device descriptor reserved for Device Configuration variables (the DevCon section). There is one standard field with a length of 32 bytes in the DevCon section. This is the player default character set. If this field is all 0's, it is considered to be the CD-I default character set (ISO 8859-1). Otherwise, this field will contain the ISO 2022 escape sequence for the player default character set.

## VII.3 Device Drivers

## 3.1.4.2 CDFM Path Descriptors

Every open path is represented by a data structure called a path descriptor. Every time an `I$Open` system call is invoked, a path descriptor is created. It contains information required by file managers and device drivers to perform I/O functions. Path descriptors are dynamically allocated and de-allocated as paths are opened and closed.

Path descriptors have three sections, a standard section, a file manager section, and an options section.

## 3.1.4.2.1 Standard Path Descriptor Fields

The first 42 bytes of the Path Descriptor are defined in the same way for all path descriptors in CD-RTOS. These fields may be read by the drivers, but may not be modified by them.

Figure VII.31 Standard Path Descriptor Fields

Offset	Length	Name	Description
0	2	PD_PD	Path number
2	1	PD_MOD	Mode (read/write/update)
3	1	PD_CNT	Number of paths using this P.D.
4	4	PD_DEV	Device Table entry address
8	2	PD_CPR	Current process ID
10	4	PD_RGS	Caller's register stack pointer
14	4	PD_BUF	Buffer address
18	4	PD_USER	Group/User ID of path's creator
22	4	PD_Paths	Linked list of open paths on device
26	2	PD_COUNT	Actual number of paths using this PD
28	2	PD_LProc	Last active process ID
30	12	-	Reserved

## VII.3 Device Drivers

## 3.1.4.2.2 File Manager Path Descriptor Fields

CDFM defines the following fields for its own use. Drivers will read some of these fields but normally do not write them.

Figure VII.32 CD File Manager Path Descriptor Fields

Offset	Length	Name	Description
42	2	PD_BIn	Number of characters in the File Manager buffer.
44	4	PD_Cp	Current File position.
48	4	PD_FP	Actual sector position of beginning of file.
52	2	PD_INTLV	File interleave factor.
54	4	PD_Siz	File size.
58	4	PD_FDAddr	Byte address of file descriptor.
62	4	PD_NxtPD	Next path descriptor in Play list.
66	4	PD_LstPD	Last path descriptor in Play list.
70	4	PD_PSRT	Starting sector of Play selection. This is the logical starting sector of the Play selection.
74	2	PD_PROCID	Process ID of Owner of this path. Signals from the Play Control List and Play Control Block are sent to the process with this ID.
76	4	PD_PCBptr	Pointer to Play Control Block or Status Block
80	2	PD_DIRECT	Direction of head movement.
82	4	PD_PCB	Reserved for CDFM use.
86	4	PD_PCL	Reserved for CDFM use.
90	4	PD_PCLArr	Reserved for CDFM use
94	4	PD_DTPtr	Drive Table Pointer
98	6		Reserved
104	1	PD_FNum	File number.
105	1	PD_CDFlags	CDFM flag bits

## VII.3 Device Drivers

## 3.1.4.2.3 Option Table Path Descriptor

The first fields of the option table are copied directly from the device descriptor initialization fields.

Figure VII.33 Option Table of a CDFM Path Descriptor

Offset	Length	Name	Description
128	1	PD_DTP	Device class
129	1	PD_CDFC	Function class
130	1	PD_ErrSz	Number of bytes per error bit
131	1	PD_NDscs	Number of discs in player
132	1	PD_DNum	Device number for Audio Processor or Device number for CD device
133	1	PD_BFctr	Blocking factor
134	4	PD_Did	Disc ID
138	2	PD_Track	Track type. This indicates the type of track (i.e. CD-I, CD-DA).
140	2	PD_XAR	Extended attribute record size.
142	4	PD_CNUM	Channel mask. This may be changed by the user at any time.
146	32	PD_TOK	File name. This is the file name of the open path.
178	2	PD_PTNUM	Path Table Entry Number
180	2	PD_PTOFF	Path Table Entry Offset

## VII.3 Device Drivers

## 3.1.4.3 CDFM Device Driver Static Storage

CDFM device driver modules contain a package of subroutines that perform sector oriented I/O to, or from, a specific hardware controller. Because these modules are re-entrant, one copy of the module can simultaneously run several identical I/O controllers.

The kernel will allocate a static storage area for each device (which may control several drives). The size of the storage area is given in the device driver module header (M\$Mem). Some of this storage area is required by the kernel and CDFM. The device driver may use the remainder in any manner.

Figure VII.34 CDFM Device Driver Static Storage

Offset	Length	Name	Description
0	4	V_PORT	Device base port address
4	2	V_LPRC	Last active process ID. This contains the process ID of the last process to use the device.
6	2	V_BUSY	Not used by CDFM. See V_Play
8	2	V_WAKE	Process ID to awaken. This contains the process ID of any process that is waiting for the device to complete I/O (0 = no process waiting). Maintained by the device driver.
10	4	V_PATHS	Linked list of open paths. This is a singly-linked list of all paths currently open on this device. It is maintained by the kernel.
14	32		Reserved
46	1	V_NDRV	Number of drives. This contains the number of drives that the controller can use. It is defined by the device driver as the maximum number of drives that the controller can work with. CDFM will assume that there is a drive table for each drive.
47	1	V_NAP	Number of Audio Processors
48	6		Reserved
54			Drive tables. This contains one table per drive that the controller will handle. CDFM will assume there are as many tables as specified in V_NDRV.

## VII.3 Device Drivers

## 3.1.4.3.1 Drive Tables

After the driver's INIT routine has been called, CDFM requests the driver to initialize the drive table for the corresponding drive. As mentioned in Figure VII.34 (last item), each drive has a corresponding drive table. The drive table format is as follows:

Figure VII.35 CDFM Drive Table Format

Offset	Length	Name	Description
0	4	V_LastPos	Last seek position. Initialized by the CDFM device driver
4	4	V_PTSiz	Path table size. Initialized by the CDFM
8	4	V_PTAdd	Path table address. Handled only by the CDFM. Set to zero by the driver in INIT routine
12	2	V_CHILD	Child process number
14	4	V_PLAY	Pointer to path descriptor of Play path. Initialized by the CDFM and the CDFM device driver.
18	4	V_DIRCT	Direction of head movement. Initialized by the CDFM
22	4	V_DSize	Size of CD-I disc. Initialized by the CDFM
26	4	V_ChanMask	Channel mask. Initialized by the CDFM device driver
30	4	V_APMask	Audio channel mask. Initialized by the CDFM device driver
34	1	V_PlayFlag	Play in progress flag. Initialized by the CDFM device driver
35	1	V_Paused	Drive paused flag. Initialized by the CDFM device driver
36	1	V_BFctr	Blocking factor. Initialized by the CDFM
37	1	V_IRQlv	Hardware interrupt level. Initialized by the CDFM device driver

(continued)

Figure VII.35 FM Drive Table Format (continued)

Offset	Length	Name	Description
38	1	V_ROMFlag	CDROM disc flag. Initialized by the CDFM device driver
39	1	V_PTVValid	Path Table valid flag. Initialized by the CDFM. Set to zero when the driver detects disc change
40	4	V_SoundMap	Pointer to current playing soundmap descriptor by the CDFM device driver
44	2	V_ASigPrc	Process identifier for audio signal. Initialized by the CDFM device driver
46	1	V_AudOff	Flag for turning off audio. Initialized by the CDFM device driver
47	1	V_AudPlay	Flag for soundmap output. Initialized by the CDFM device driver
48	1	V_ErFlags	Error handling flags
49	1		Reserved
50	2	V_SMPPath	Path on which current soundmap was started
52	4	V_DirSect	Directory sector buffer address
56	4	V_DirSectNum	Directory sector number
60	20		Reserved

The V\_PlayFlag field indicates the type of play in progress:

Figure VII.36 V\_PlayFlag Field

Value	Name	Description
1	Play_CDI	SS_Play or I\$Read of CD-I disc in progress
2	Play_Seek	SS_Seek in progress
3	Play_CDDA	SS_CDDA in progress
4	Play_Read	I\$Read of CD-ROM Mode 1 disc in progress



### VII.3 Device Drivers

---

#### 3.1.5 CDFM Device Driver Subroutines

As with all CD-RTOS device drivers, CDFM device drivers use a standard executable memory module format with a module type code "Drivr".

The execution offset address in the module header points to a branch table that has eight entries. Each entry is the offset to a corresponding subroutine. The branch table is as follows:

Diskent	dc.w INIT	Initializes device
	dc.w READ	Reads character
	dc.w WRITE	(Not used)
	dc.w GETSTAT	Gets device status
	dc.w SETSTAT	Sets device status
	dc.w TERM	Terminates device
	dc.w TRAP	Trap entry point
	dc.w PLAY	Plays selection

Each subroutine should exit with the condition code register carry bit cleared, i.e. set to zero, if no error occurred. Otherwise the carry bit should be set to one and an appropriate error code returned in d1.w.

VII.3 Device Drivers

---

**3.1.5.1 The INIT Subroutine**

The INIT routine's function is to initialize the device and the associated static storage. The INIT routine must:

- (1) Initialize the device's permanent storage. This minimally consists of initializing V\_NDRV to the number of drives with which the controller will work.
- (2) Initialize device control registers and enable interrupts.
- (3) Place the IRQ service routine on the IRQ polling list by using the F\$IRQ service request.

**Note:** Prior to being called, the device permanent storage will be cleared (set to zero) except for V\_PORT which will contain the device address. The driver should initialize each drive table appropriately for the type of disc the driver expects to be used on the corresponding drive.

Input:           (a1) = Device descriptor pointer  
                   (a2) = Device static storage pointer  
                   (a4) = Current process descriptor pointer  
                   (a6) = System global data storage pointer

Output:           None

Error Output:   cc    = Carry bit set to one  
                   d1.w = Error code

**3.1.5.2 The READ Subroutine**

The function of the READ subroutine is to read 'Mode 1' blocks from the disc.

Input:           d2.l = Disc logical sector number to read  
                   d3.l = Number of sectors to read  
                   (a0) = Read buffer pointer  
                   (a1) = Path descriptor pointer  
                   (a2) = Device static storage pointer  
                   (a4) = Current process descriptor pointer  
                   (a6) = System global data storage pointer

Output:           None

Error Output:   cc    = Carry bit set to one  
                   d1.w = Error code

### VII.3 Device Drivers

---

#### 3.1.5.3 The WRITE Subroutine

This subroutine is currently illegal on CD-I media. Any attempt to use the WRITE subroutine will return with the carry bit set to one and an error code of E\$UnkSvc in d1.w.

#### 3.1.5.4 The GETSTAT and SETSTAT Subroutines

These two subroutines are grouped together because of their similarity in use. These routines are used to get/set the device's operating parameters as specified for the I\$GetStt and I\$SetStt service requests.

When the driver's getstat or setstat entry point is called, the registers have the following values:

Input:	d0.w = Function code (a1) = Path descriptor pointer (a2) = Device static storage pointer (a4) = Current process descriptor pointer (a5) = Pointer to user's register stack (a6) = System global data storage pointer Other registers are function code dependent
Output:	Function code dependent
Error Output:	cc = Carry bit set to one d1.w = Error code

### VII.3 Device Drivers

---

The following additional input registers are used for the CDFM functions listed:

**SS\_Raw**

- d2.l = Disc logical sector number to read
- d3.l = Number of 2340 bytes sectors to read
- (a0) = Address of read buffer

**SS\_ReadTOC**

- d2.l = Number of bytes
- (a0) = Address of Buffer

**SS\_Seek**

- d2.l = Logical sector number to seek to
- (a0) = Pointer to status block

**SS\_CDDA**

- d2.l = Logical sector number to start play
- d3.l = Number of sectors to play
- (a0) = Pointer to status block

**SS\_Mount**

- d2.w = Disc number

### 3.1.5.5 The TERMINATE Subroutine

This routine is called when a device is no longer in use in the system. This is defined as when the link count of its device table entry becomes zero.

The TERM subroutine must:

- (1) Wait until any pending I/O has completed.
- (2) Disable the device interrupts.
- (3) Remove the device from the IRQ polling list.

Input:           (a1) = Device descriptor pointer  
                  (a2) = Device static storage pointer  
                  (a4) = Current process descriptor pointer  
                  (a6) = System global data pointer

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

### 3.1.5.6 The TRAP subroutine

The TRAP entry should be defined as the offset to the exception handling code or zero if no handler is available. This entry point is currently not used by the kernel.

### 3.1.5.7 The PLAY Subroutine

The PLAY subroutine sets up the controller's registers to 'play' a selection. The PLAY routine must:

- (1) Get the starting sector for the play from the path descriptor.
- (2) Seek to the starting sector if necessary.
- (3) Set up the file number selection mechanism using the file number from the path descriptor.
- (4) Set up the channel number selection mechanism using the channel mask from the Play Control Block.

**Note:** The Play entry point will be called to read all Mode 2 sectors. The Read entry will be called to read Mode 1 sectors.

Input:           (a1) = Path descriptor pointer  
                  (a2) = Device static storage pointer  
                  (a4) = Process descriptor pointer  
                  (a6) = System global data storage pointer

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

### 3.1.5.8 The IRQ Service Request Subroutine

Although this routine is not included in the device driver module branch table and is not called directly by the CDFM, it is a key routine in interrupt driven device drivers. Its general functions include:

- (1) Poll the device to determine the interrupt type. If the interrupt is not caused by the CD controller, the carry bit must be returned set to one with an RTS instruction as quickly as possible.
- (2) Service device interrupts.
- (3) When the IRQ service routine finishes servicing an interrupt it must clear the carry bit (i.e. set to zero) and exit with an RTS instruction.

The IRQ routine must perform very specific instructions when servicing a PLAY interrupt. These functions are as follows:

- (1) Determine the type of Coding Information (Audio, Video, or Data).
- (2) Get the path descriptor pointer out of V\_PLAY.
- (3) Get the PCB out of the path descriptor.
- (4) Get the PCL for the type of Coding Information.
- (5) Update the PCL and PCB counts.
- (6) Copy the data into the appropriate PCB buffer (audio processor, video processor, or memory).
- (7) Send the PCL\_Sig if the buffer for the PCL is full.
- (8) Determine if the Play is finished. If so, send the PCB\_Sig signal.
- (9) Get the next Play path descriptor from PD\_NxtPD.
- (10) Put the next Play path descriptor in V\_PLAY.
- (11) Wake up the next PLAY process.

Input:                   (a2) = Device Static storage pointer  
                              (a3) = Port address  
                              (a6) = System global data storage pointer

**Note:** IRQ service routines may destroy the following registers only:

d0, d1, a0, a2, a3 and a6.

All other registers must be preserved.



### 3.2 UCM Driver Interface

This section defines the interface between the UCM and the drivers which interface it to the user input-output hardware.

#### 3.2.1 Driver and Descriptor Organization and Initialization

The UCM manages three types of I/O; video display output, pointer input and keyboard input. However, the functions of these devices are often inter-related. Therefore, a mechanism is provided to logically tie these devices together, thereby creating one logical path to these devices. UCM requires that the video device descriptor references the pointer device and the keyboard device if one is available.

This enables the application to reference all devices on a single path and enables the system to work more closely with the devices. The pointer and keyboard descriptors should only reference themselves.

When an application issues an `I$Open` service request, the CD-RTOS kernel allocates a path descriptor for the path and issues an `I$Attach` call for the device named in the service request. The `I$Attach` function will allocate static storage for the device and call the device driver's `Init` routine if this is the first `I$Attach` issued for the device.

The kernel then calls the UCM's `Open` subroutine. The UCM checks the `PD_D2N` field of the device descriptor to see if an additional device is named. If not, UCM returns. If so, the UCM issues an `I$Attach` service request for the device named. UCM then repeats this process for the device named in `PD_D3N`.

## VII.3 Device Drivers

## 3.2.2 Data Structures

## 3.2.2.1 Device Descriptor Format

Device descriptor modules are non-executable modules that provide information which associates I/O devices with their logical names, hardware controller address(es), device driver name, file manager name, and initialization parameters.

## 3.2.2.1.1 Standard Device Descriptor Fields

The first 72 bytes of each device descriptor are defined in the same way for all device descriptors in CD-RTOS. For further information refer to A VII.1.

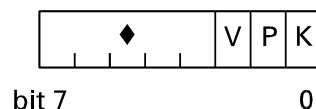
## 3.2.2.1.2 Device Descriptor Option Fields

The options fields of the device descriptor describe the device dependent options for a device. They are copied into and referenced from the path descriptor when a process opens a path to the device.

Figure VII.37 UCM Device Descriptor Option Fields

Offset	Length	Name	Description
72	1	PD_DTP	Device type (6 for UCM)
73	27		Reserved
100	1	PD_FC	Classes of functions of which device is capable
101	1		Reserved
102	2	PD_D2N	Offset to second device name
104	2	PD_D3N	Offset to third device name
106	2		Reserved

The format of the field PD\_FC is as follows.



V = Capable of video display output

P = Capable of pointer input

K = Capable of keyboard input

## VII.3 Device Drivers

## 3.2.2.2 Path Descriptor Format

Every time an `I$Open` system call is invoked, a path descriptor is created. It is used to store information required by the kernel, the file manager, and the driver to perform an I/O function. Path descriptors have three sections, a standard section, a file manager section, and an options section.

## 3.2.2.2.1 Standard Fields

The first forty-two bytes of each path descriptor are defined in the same way for all path descriptors in CD-RTOS. These fields may be read by the drivers but they may not modify them. For more information, see VII.3.1.4.2.1.

## 3.2.2.2.2 File Manager Fields

The UCM defines the following fields for its own use. Drivers may read these fields but they may not write to them.

Figure VII.38 UCM File Manager Fields

Offset	Length	Name	Description
42	4	Reserved	Space for PD_DV2 of SCF
46	2	Reserved	Space for PD_MAX of SCF
48	1	Reserved	Space for PD_RAW of SCF
49	1	PD_DFLGS	Device detach flags
50	4	PD_KDEV	Keyboard device table entry
54	4	PD_KSTAT	Keyboard driver static storage
58	4	PD_PDEV	Pointer device table entry
62	4	PD_PSTAT	Pointer driver static storage
66	4	PD_VDEV	Video device table entry
70	4	PD_VSTAT	Video driver static storage
74	4	PD_FEXEC	Driver routine to execute
78	4	PD_EXTMOD	External Subroutine Module Pointer
82	4	PD_ChOtData	Character output data area pointer
86	4	PD_EModData	External Subroutine Module data area

## VII.3 Device Drivers

The UCM also defines the following fields in the Character Output Data Area, pointed to by PD\_ChOtData. These fields are to be used by the video device driver for maintaining the status of the I\$Write and I\$WriteLn functions; field PD\_Flags is also used by UCM. These fields are written to only by the device driver.

Figure VII.39 UCM File Manager Video Driver Fields

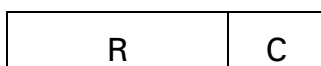
Offset	Length	Name	Description
0	2	PD_LastRow	Highest row number <sup>25</sup> for character output
2	2	PD_LastCol	Highest column number <sup>26</sup> for character output
4	2	PD_CurX	Horizontal cursor <sup>26</sup> location
6	2	PD_CurY	Vertical cursor <sup>25</sup> location
8	2	PD_Flags	Character output flags
10	4	PD_COMD	Pointer to character output drawmap descriptor
14	2	PD_Beg0	Beginning glyph number for 1st active font
16	2	PD_End0	Ending glyph number for 1st active font
18	4	PD_AFDData0	Pointer to beginning of 1st active font data
22	2	PD_Beg1	Beginning glyph number for 2nd active font
24	2	PD_End1	Ending glyph number for 2nd active font
26	4	PD_AFDData1	Pointer to beginning of 2nd active font data
30	2	PD_Beg2	Beginning glyph number for 3rd active font
32	2	PD_End2	Ending glyph number for 3rd active font
34	4	PD_AFDData2	Pointer to beginning of 3rd active font data
38	2	PD_Beg3	Beginning glyph number for 4th active font
40	2	PD_End3	Ending glyph number for 4th active font
42	4	PD_AFDData3	Pointer to beginning of 4th active font data
46	1	PD_Byte	Save byte for write

<sup>25</sup> In lines, starting from 0

<sup>26</sup> In pixels, starting from 0

VII.3 Device Drivers

UCM uses the least significant two bits of the PD\_Flags field to determine how it to process characters on the I\$WriteLn function. These bits must be set by the video driver when it executes the CO\_SCMM function. The format of the PD\_Flags field follows:



- R - Reserved for driver use
- C - Character output mapping method
  - 00 = Seven-bit
  - 01 = Seven/Fifteen-bit
  - 10 = Sixteen-bit
  - 11 = Reserved

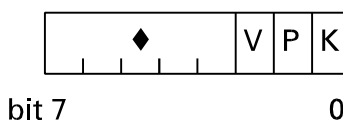
3.2.2.2.3 Options Fields

All options fields of the device descriptor are copied to the options fields of the path descriptor.

Figure VII.40 UCM Option Fields of a Path Descriptor

Offset	Length	Name	Description
128	1	PD_DTP	Device type (6 for UCM)
129	27		Reserved
156	1	PD_FC	Classes of functions of which device is capable
157	1		Reserved
158	2	PD_D2N	Offset to second device name
160	2	PD_D3N	Offset to third device
162	2	PD_CNum	Current column number
164	1	PD_Errs	Most recent I/O error
165	3		Reserved

The format of the field PD\_FC is as follows.



- V = Capable of video display output
- P = Capable of pointer input
- K = Capable of keyboard input

## VII.3 Device Drivers

## 3.2.2.3 Driver Static Storage Format

The first 46 bytes of the driver static storage are defined by CD-RTOS for all drivers:

Figure VII.41 Standard Driver Static Storage Fields

Offset	Length	Name	Description
0	4	V_PORT	Device base port address
4	2	V_LPRC	Last active process ID
6	2	V_BUSY	Current process ID (0 = unbusy)
8	2	V_WAKE	Active process ID if driver must wake up
10	4	V_Paths	Linked list of open paths on device
14	32	-	Reserved

UCM also requires variables in the static storage area. In addition to some reserved area, UCM requires pointers to the Read, Write, GetStat and SetStat routines of the device driver. UCM requires the device driver to initialize these fields when the driver's Init routine is called. The definition of the UCM fields follows:

Figure VII.42 UCM Driver Static Storage Format

Offset	Length	Name	Description
46	16		Reserved
62	4	V_READ	Address of Read routine
66	4	V_WRIT	Address of Write routine
70	4	V_PSTA	Address of SetStat routine
74	4	V_GSTA	Address of GetStat routine
78	4	V_EXTMOD	Address of subroutine extension module
82	80	V_EXMOD	Reserved data area for subroutine extensions module
162	4	V_EXTLNK	Link count of subroutine extension module
166	28		Reserved
194		V_UCM	Start driver static storage

### 3.2.3 Driver Functions

This section describes the entry points that each driver will contain, the functions they are to perform, the parameters that are passed, and any return values that must be provided. All UCM drivers, like all other CD-RTOS drivers, have the seven standard entry points defined in Appendix VII.1. These are Init, Read, Write, GetStat, SetStat, Term and Error. When necessary, the driver adds a device interrupt service routine to the CD-RTOS interrupt polling system.

#### 3.2.3.1 Init

The Init function is called when the device is first used. This function must be implemented in all UCM drivers. It should initialize any static storage variables, allocate any additional storage, initialize the device, and set up any necessary interrupts.

The driver is also required by UCM to initialize the V\_READ, V\_WRIT, V\_PSTA and V\_GSTA fields of the driver static storage to point to the driver Read, Write, SetStat and GetStat functions, respectively.

Input:	(a1) = Device descriptor pointer (a2) = Device static storage pointer (a4) = Current process descriptor pointer (a6) = System global data pointer
Output:	None
Error Output:	cc = Carry bit set to one d1.w = Error code
Possible Errors:	F\$IRQ, F\$Event, F\$SRqCMem, E\$IdFull, F\$SRtMem, E\$Unit, E\$BMode, F\$SRqMem

VII.3 Device Drivers

---

**3.2.3.2 Read**

The UCM keyboard input driver is required to implement this function. The Read function returns the next available character from the keyboard input buffer. If no character is available, then the driver should wait until a character is available.

Input:           (a1) = Path descriptor pointer  
                  (a2) = Device static storage pointer  
                  (a4) = Current process descriptor pointer  
                  (a6) = System global data pointer

Output:           d0.b = Input character

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$Read, E\$NotRdy



### 3.2.3.3 Write

The UCM video driver must implement the Write function. This entry point provides the function where drawmaps can be used to emulate the display output function of CRT type terminals. In addition to displaying characters, Write must also perform all of the control code functions defined in VII.2.3.4.9.

Input:           d0.w = Path number  
                  d1.l = Number of bytes to write  
                  (a0) = Pointer to characters  
                  (a1) = Path descriptor pointer  
                  (a2) = Device static storage pointer  
                  (a4) = Current process descriptor pointer  
                  (a6) = System global data pointer

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: E\$NoFont, E\$NoDM

VII.3 Device Drivers

---

**3.2.3.4 GetStat and SetStat**

The GetStat and SetStat driver subroutines are used to implement all of the extra functions that are specified in VII.2.3. The general UCM functions are defined in VII.2.3.3, the video driver's functions are defined in VII.2.3.4, the keyboard driver's in VII.2.3.6, and the pointer driver's in VII.2.3.5. The SS\_SLink function is handled completely at UCM level, for the other functions the UCM passes the parameters for each function to driver in exactly the same way as defined in VII.2.3. Any return values are also the same. In addition, the parameters defined in the input list below also are always passed to the GetStat and SetStat routines. If the function code or getstat/setstat code is unknown, error E\$UnkSvc is returned.

Input:           d0.w = System path number  
                 (a1) = Path descriptor pointer  
                 (a2) = Device static storage pointer  
                 (a4) = Current process descriptor pointer  
                 (a5) = Pointer to user's registers  
                 (a6) = System global data pointer

Output:           Function dependent

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: Function dependent

VII.3 Device Drivers

---

**3.2.3.5 Term**

The Term subroutine is called when the device is no longer in use. This function is required by all UCM drivers. It should clean up, disabling interrupts and events set-up, unlinking linked modules, returning any allocated storage and putting devices into an "off" state.

Input:           (a1) = Device descriptor pointer  
                  (a2) = Device static storage pointer  
                  (a4) = Current process descriptor pointer  
                  (a6) = System global data pointer

Output:           None

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

Possible Errors: F\$IRQ, F\$Event, F\$SRtMem, F\$UnLink

### 3.2.3.6 The TRAP subroutine

The TRAP entry should be defined as the offset to the exception handling code or zero if no handler is available. This entry point is currently not used by the kernel.

VII.3 Device Drivers

---

**3.2.3.7 IRQ**

Although this routine is not included in the device driver module branch table and is not called directly by the UCM, it is a key routine in interrupt driven device drivers. Its general functions include:

1. Poll the device to determine the interrupt type. If the interrupt was not caused by the device, the driver must set the carry bit to one and exit with an RTS as quickly as possible.
2. Service device interrupts.
3. When the IRQ service routine finishes servicing an interrupt it must clear the carry bit (i.e. set to zero) and exit with an RTS instruction.

Input:           (a2) = Device static storage pointer  
                  (a3) = Device base port address  
                  (a6) = System global data pointer

**Note:** IRQ service routines may destroy the following registers only:

d0, d1, a0, a2, a3 and a6.

All other registers must be preserved.

### 3.3 NRF Driver Interface

This section defines the interface between the NRF and its drivers.

#### 3.3.1 General

A device driver is required for the NRF file manager to handle the actual initialization of the NVRAM area the first time the NVRAM is accessed at the time of the first machine startup and to read and write bytes from/to the NVRAM.

The read/write routines are required to be part of the device driver because the physical NVRAM area need not be fully contiguous RAM. The NVRAM area could consist of only the even or odd addresses of the NVRAM space, or it could be fully contiguous RAM of the full address space. Because of this, NRF only deals with logical addresses of the NVRAM area and the device driver must translate this logical address into a physical memory address. The first logical byte of the NVRAM area is 0 while the first physical byte of the NVRAM area is the port address of the NVRAM given in the device descriptor.

VII.3 Device Drivers

---

**3.3.2 Device Descriptor Format****3.3.2.1 Standard Device Descriptor Fields**

The first 72 bytes of the device descriptor are defined in the same way for all device descriptors in CD-RTOS, see Appendix VII.1. The M\$Port field of the device descriptor should contain the base address of the NVRAM area.

**3.3.2.2 Device Descriptor Option Fields**

The initialization table of the device descriptor is copied into part of the descriptor option section when a path to the device is opened.

Figure VII.43 **Device Descriptor Option Fields**

Offset	Length	Name	Description
72	1	PD_DTP	Device type
73	1	PD_CONT	Contiguous Flag, 0 = non-contiguous 1 = contiguous
74	2	PD_RSIZ	RAM size for NVRAM

### 3.3.3 Path Descriptor Format

#### 3.3.3.1 Standard Fields

The first forty-two bytes of each path descriptor are defined in the same way for all path descriptors in CD-RTOS. These fields may be read by the drivers but they may not be modified by them. For more information, refer to Appendix VII.1.2.

#### 3.3.3.2 File Manager Fields

NRF defines the following fields for its own use. Drivers may read these fields but not modify them.

Figure VII.44 File Manager Fields

Offset	Length	Name	Description
42	4	PD_CP	Current file position
46	4	PD_BLKLA	Block header logical address
50	4	PD_STAT	Pointer to device static storage
54	4	PD_HDBUF	Block header buffer pointer

#### 3.3.3.3 Option Fields

The following fields are defined as the options section (PD\_OPT) of the path descriptor. The first two fields are copied from the device descriptor initialization table, the last field is set up when a file is opened or created.

Figure VII.45 Option Fields

Offset	Length	Name	Description
128	1	PD_DTP	Device type
129	1	PD_CONT	Contiguous NVRAM flag
130	2	PD_RSIZ	NVRAM size
224	32	PD_NAM	File name



## VII.3 Device Drivers

## 3.3.4 Driver Static Storage

## 3.3.4.1 Logical Format of NVRAM Area

At logical address 0 of the NVRAM is a sync long word which is used as a flag to indicate that the NVRAM has been initialized. At logical address 4 of NVRAM is the logical address of the free block.

Figure VII.46 NVRAM Logical Addresses

LogAddr	Length	Description
0	4	Sync bytes (\$dead code)
4	4	Logical Address of Free Block

The NVRAM area is managed by NRF as a list of linked file blocks. Each file has a logically contiguous block of NVRAM which includes the block header and the actual data storage area for the file. The free space is kept as a contiguous free block which shares the same header as a standard file, it is identified by having the name field start with a 0xFF character.

Figure VII.47 NVRAM File Manager

Offset	Length	Name	Description
0	28	NVF_FILNAME	File name
28	4	NVF_LSN	LSN for RBF compat. (address of FD info)
32	4	NVF_NEXTBLK	Logical address of next block
36	4	NVF_PREVBLK	Logical address of previous block
40	4	NVF_BLKSIZE	Size of block, including header
44	1	NVF_RES1	Reserved byte
45	1	NVF_FDATT	File attributes (start of FD info)
46	2	NVF_FDOWN	File owner
48	5	NVF_FDDATE	Last modified data (YYMMDDHHMM)
53	1	NVF_FDLINK	Link count
54	4	NVF_FDSIZE	File data size
58	3	NVF_FDDCR	Date created (YYMMDD)
61	1	NVF_RES2	Reserved byte

### VII.3 Device Drivers

---

#### 3.3.5 Driver functions

This section describes the entry points that the driver will contain, the functions they are to perform, the parameters that are passed and any return values that must be provided. All NRF drivers have the seven standard entry points defined in Appendix VII.1, these are Init, Read, Write, GetStat, SetStat, Term and IRQ.

##### 3.3.5.1 Init

The Init function is called when the NVRAM is first used. This function should be implemented to check that the NVRAM area has been initialized and if not initialize it with a free block size of the NVRAM area.

The parameters for the Init routine are as follows:

Input:	(a1) = Device descriptor pointer
	(a2) = Device static storage pointer
	(a4) = Current process descriptor pointer
	(a6) = System global data pointer
Output:	None
Error Output:	cc = Carry bit set to one
	d1.w = Error code

VII.3 Device Drivers

---

**3.3.5.2 Read**

The Read routine is required to read the given number of bytes from the NVRAM area into the data buffer. A driver Read routine is required so that the physical NVRAM area may be non-contiguous addresses, such as using only the odd or even addresses of the NVRAM address space.

Input:           d0.1 = Number of bytes to read  
                  d2.1 = Logical address of NVRAM to read  
                  (a1) = Path descriptor pointer  
                  (a2) = Device static storage pointer  
                  (a4) = Current process descriptor pointer  
                  (a6) = System global data pointer

Output:           byte(s) returned in buffer given by PD\_BUF

Error Output:    cc    = Carry bit set to one  
                  d1.w = Error code

### 3.3.5.3 Write

The Write routine is used to write bytes to the NVRAM area.

Input:	d0.1 = Number of bytes to write
	d2.1 = Logical address of NVRAM to write
	(a1) = Path descriptor pointer
	(a2) = Device static storage pointer
	(a4) = Current process descriptor pointer
	(a6) = System global data pointer
Output:	byte(s) written to NVRAM from PD_BUF
Error Output:	cc = Carry bit set to one
	d1.w = Error code

#### 3.3.5.4 **GetStat and SetStat**

The NRF file manager directly handles all the Get/Set Stat codes defined in VII.2.4.9, any other code sent to the file manager will be passed directly to the driver with the following parameters:

Input:	d0.w = Function code
	(a1) = Path descriptor pointer
	(a2) = Device static storage pointer
	(a4) = Current process descriptor pointer
	(a6) = System global data pointer
Output:	Function dependent
Error Output:	cc = Carry bit set to one
	d1.w = Error code

### 3.3.5.5 Term

The Term routine is called when the device is no longer in use. This function can just return with no error, it need not do anything.

Input:           (a1) = Device descriptor pointer  
                  (a2) = Device static storage pointer  
                  (a4) = Current process descriptor pointer  
                  (a6) = System global data pointer

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

### 3.3.5.6 The TRAP subroutine

The TRAP entry should be defined as the offset to the exception handling code or zero if no handler is available. This entry point is currently not used by the kernel.

## VII.4 Synchronization Primitives

---

### VII.4 Synchronization Primitives

CD-RTOS provides several low-level mechanisms for use by an application program to synchronize e.g. visual effects with audio data. The most basic mechanism available is to simply rely on the inherent ordering of blocks of data on disc. In some applications (e.g., slide show presentations), the ordering of data may provide all the synchronization necessary.

A simple improvement can be made by reading the system clock, and executing a function at a particular time. CD-RTOS provides the time of day in increments of 1/100th of a second for this case.

More complicated synchronization methods are possible using an event and/or signals (e.g. software interrupts). Events are similar to semaphores and are described completely in the CD-RTOS technical manual (Appendix VII.1). The use of signals are covered here. Signals may be generated for a process in any of the following ways:

- The presence of a trigger (T), end of record (EOR), or end of file (EOF) bit in the subheader submode byte of a real-time sector (see SS\_Play in VII.2.2.3.2).

**Note:** (1) The EOF and T bits should generate interrupts only after file number selection.

(2) The EOR bit should generate an interrupt only after the file and channel number selection.

- The buffer of a play control list becoming full (see SS\_Play in VII.2.2.3.2).
- The expiration of a software timer (e.g., from the F\$Alarm system call see VII.1.1.4.2).
- The receipt of data from a user input device (see I\$SetStt PT\_SSig in VII.2.3.5 or KB\_SSig in VII.2.3.6).
- A software interrupt (signal) sent from other concurrent processes.
- The video line display interrupt.

If a process makes a call to PT\_SSig, KB\_SSig or DC\_SSig while another signal request is pending, the call will return the error E\$NotRdy. After receiving the signal from one of these calls the application must "re-arm" the signal by calling the function again in order to receive another signal.



Table of Contents

---

**Chapter VIII Base Case System**

	<b>Page</b>	
VIII.1	Introduction	VIII-1
	1.1 General	VIII-1
	1.2 Base Case Model	VIII-2
	1.3 System Module Names	VIII-5
VIII.2	CD-I Disc Format	VIII-6
	2.1 General	VIII-6
	2.2 CD-DA	VIII-6
	2.3 Mode 1	VIII-6
	2.4 EDC/ECC	VIII-6
	2.5 Sector Data Process Delay	VIII-6
VIII.3	Data Retrieval Structure	VIII-7
	3.1 General	VIII-7
	3.2 ISO 9660 File Structure	VIII-7
VIII.4	Base Case Audio	VIII-8
	4.1 General	VIII-8
	4.2 Audio Mixing Control	VIII-8
	4.2.1 Base Case System Audio Source	VIII-8
	4.2.2 External Audio Source	VIII-8
	4.3 Audio Decoder Delay	VIII-9
	4.4 Audio Output	VIII-9
VIII.5	Base Case Video	VIII-10
	5.1 Display	VIII-10
	5.2 Color Resolution	VIII-10
	5.3 Video Functions	VIII-10
	5.4 Image Resolution	VIII-10
	5.5 Display Control Program	VIII-10
	5.6 Mosaic Pixel Repeat Factors	VIII-11
	5.7 Mattes	VIII-11
	5.8 Video Decoder Delay	VIII-11
	5.9 External Video Source	VIII-11
VIII.6	Program-Related Data Representations	VIII-12
	6.1 General	VIII-12
	6.2 Executable Object Code	VIII-12
	6.3 Character Sets	VIII-12

This page is intentionally left blank

## Table of Contents

		<b>Page</b>
VIII.7	CD-RTOS	VIII-13
	7.1 General	VIII-13
	7.2 Kernel	VIII-13
	7.3 File Managers	VIII-14
	7.3.1 CDFM I\$Setstt Functions	VIII-14
	7.3.2 CDFM File Structure Support	VIII-14
	7.4 Device Drivers	VIII-15
	7.5 Memory Usage by CD-RTOS	VIII-16
	7.5.1 CDFM Dynamic Memory Allocations	VIII-16
	7.5.1.1 Path Table and Directory Buffers	VIII-16
	7.5.1.2 I\$Read Buffering	VIII-17
	7.5.1.3 ID Tables	VIII-17
	7.5.1.4 Soundmaps	VIII-18
	7.5.2 UCM	VIII-18
	7.5.2.1 Static Storage Allocations	VIII-18
	7.5.2.2 Dynamic Memory Allocations	VIII-18
	7.5.2.2.1 ID Tables	VIII-18
	7.5.2.2.2 Drawmaps	VIII-19
	7.5.2.2.3 Regions	VIII-19
	7.5.2.2.4 FCTs and LCTs	VIII-20
	7.5.2.2.5 Fill Routines	VIII-20
	7.5.2.2.6 Polygons	VIII-20
	7.5.3 NRF	VIII-21
	7.5.4 Pipeman	VIII-21
VIII.8	System Resources and Performance	VIII-22
	8.1 General	VIII-22
	8.2 MPU	VIII-22
	8.3 RAM	VIII-22
	8.4 ROM	VIII-23
	8.5 NVRAM	VIII-23
	8.6 DMA	VIII-23
	8.7 CD-Control Unit	VIII-23
	8.8 Interrupt	VIII-23
	8.9 X-Y Device	VIII-24
	8.10 Clock-Calendar	VIII-24
	8.11 CD-Player	VIII-24

This page is intentionally left blank

## CD-I Full Functional Specification

Chapter VIII

Base Case System

### List of Illustrations

---

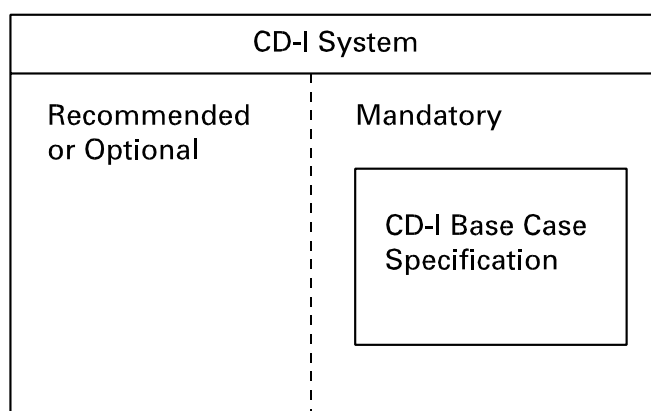
<b>Fig. no.</b>	<b>Caption</b>	<b>Page</b>
VIII.1	Specification of system	VIII-1
VIII.2	Base Case Model	VIII-3
VIII.3	Attenuator tolerances in the Base Case Audio Mixer	VIII-8

This page is intentionally left blank

**Chapter VIII Base Case System****VIII.1 Introduction****1.1 General**

A system, made up of hardware and system software, is called a CD-I system if, at the very least, it fully conforms to the CD-I Base Case Specification detailed in this chapter. Each part of the Base Case Specification is **mandatory** in any product realization of a CD-I system.

It should be noted that any system (see Figure VIII.1) that contains as a part of its **mandatory** specification the CD-I Base Case Specification, is a CD-I system if, and only if, CD-I disc based functions in this system have priority<sup>1</sup> over all other non CD-I functions when a CD-I application is active in this system.

Figure VIII.1 **Specification of system**

The specification in sections VIII.2 to VIII.7 give the **mandatory** functions that a Base Case system must support. Section VIII.8 discusses the Base Case system resource and performance specification.

---

<sup>1</sup> That is, so that no performance loss is incurred by the application running on this system relative to it having run on a CD-I Base Case System alone.

## VIII.1 Introduction

---

### 1.2 Base Case Model

The Base Case model is shown in Figure VIII.2. The CD-Control Unit, in response to commands from the operating system (i.e. CD-FM or CD-RTOS), accesses the disc via the CD-DA Controller / Decoder. When CD-DA tracks are being accessed the data is transferred directly to the Audio Processing Unit whose function is to control the attenuation, set the panning and output to analogue audio left/right signals.

If a CD-I track is being accessed the CD-Control Unit, in response to commands from the operating system (i.e. CD-FM), selects sectors from the data stream and transfers the CD-I sectors to the ADPCM decoder (for direct audio playback) or, via the system bus, to memory (for audio, video or program related data). Audio data is usually routed continuously to the ADPCM decoder which decodes it in real-time to produce analogue audio, which is output via the Audio Processing Unit. Video and program related data as well as audio data to be stored in a soundmap is routed, by means of the DMA controller or MPU, to RAM.

The RAM is organized as two separate banks each of a minimum size of 512KB, which is shared between MPU and video controller by an Access Controller. Although the amount of memory used by CD-RTOS and the system modules will vary depending upon the number of processes active and the number of open paths, it is guaranteed at start-up of the application to be less than 64k, split evenly between the two banks. All other RAM may be used for either audio, video or program related data.

Video data consists of both pixel data and video control data. The display of pixel data from RAM is by means of a two path display controller. Video data is accessed independently in the two banks of RAM, and the two display paths are combined to produce a single analogue RGB video output.

The operating system (CD-RTOS) is contained in ROM. Moreover, there is a small area of non-volatile RAM which is available both to the operating system and application programs.

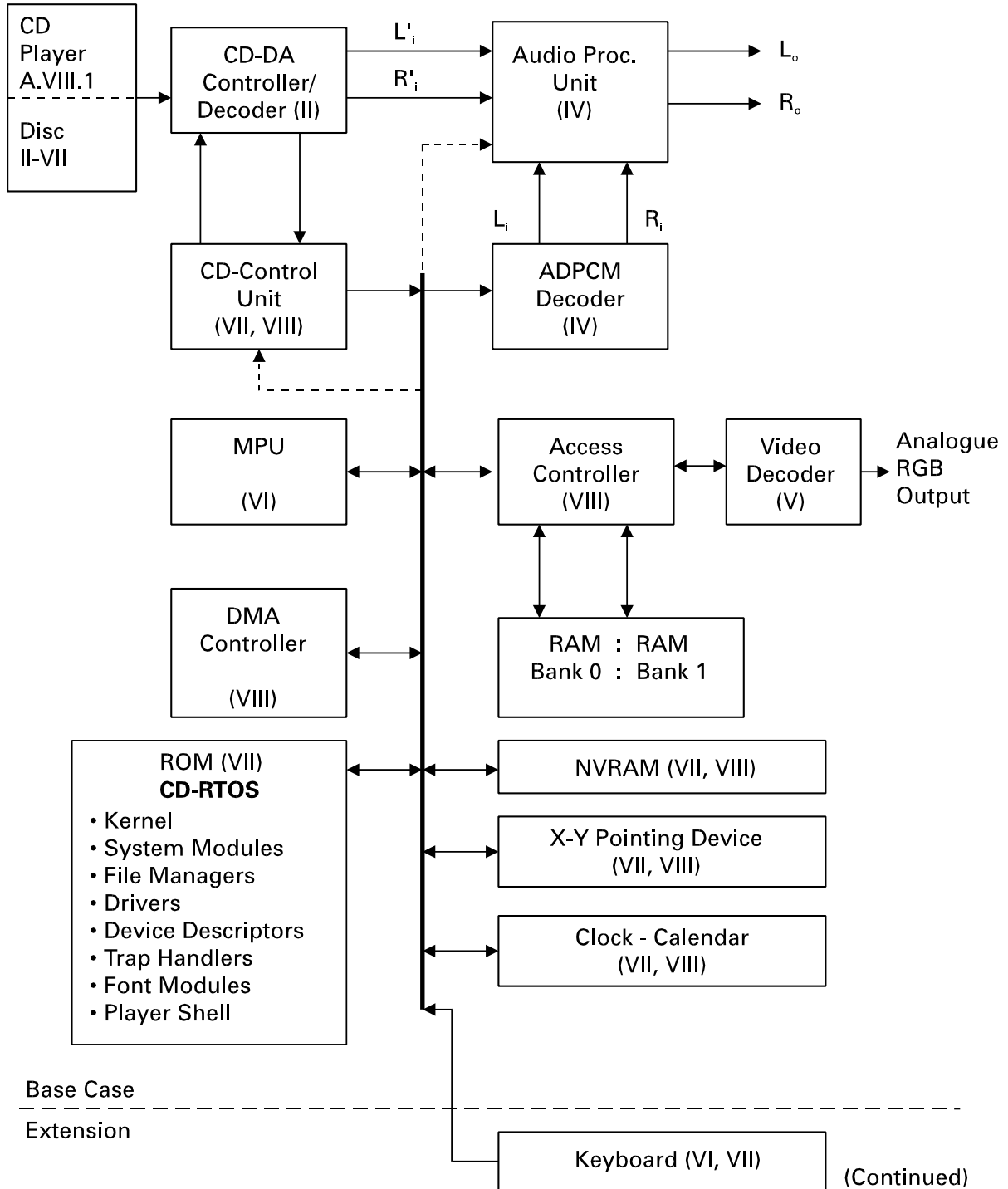
The user interface is primarily by means of an X-Y pointing device. A keyboard is supported, but does not form a part of the Base Case system. As such, it is the responsibility of any application that requires a keyboard to emulate it by means of the X-Y device and display functions if no keyboard is connected to the system. The latter is known to the application from the CSD table.

A standard character set and font module is provided in the system. Also a clock/calendar module is provided.



VIII.1 Introduction

Figure VIII.2 Base Case Model



VIII.1 Introduction

---

Figure VIII.2 **Base Case Model** (continued)

The Roman numerals indicate the relevant Chapter.

————— = 16 - bit Bus

————— = data & 'address' path

- - - - - = 'control' path

### 1.3 System Module Names

The following modules will be present in the module directory of every Base Case player. Other modules will be present for device drivers, device descriptors, the player shell and other manufacturer dependent modules. The names of these modules are not specified except that they will not begin with "cdi\_":

- kernel
- init
- csdinit
- cio
- math
- font8x8
- cdfm
- ucm
- nrf
- pipeman
- nvr

---

**VIII.2 CD - I Disc Format**

---

**VIII.2 CD-I Disc Format****2.1 General**

All parts specified in Chapter II should be supported by the Base Case System. The only additions or restrictions to this are given below.

**2.2 CD-DA**

A Base Case System must be capable of decoding CD-DA PCM data, coded in accordance with the CD-DA specifications, from either CD-DA tracks on a CD-I disc or CD-DA tracks on a CD-DA disc.

**2.3 Mode 1<sup>2</sup>**

The Base Case System should be able to read into memory sectors containing data with the mode byte (see II.4.4) value of 1 and formatted according to the Mode 1 physical format.

**2.4 EDC/ECC**

The Base Case System should be able to perform EDC/ECC for both Mode 2 Form 1 (see II.4.7.2 and II.4.7.3) and Mode 1 sectors prior to the user data being available for further processing by the system.

**2.5 Sector Data Process Delay**

The sector data processing unit (see II.6) should process data for a real-time sector (RT-bit set to one) within 27 milliseconds.

---

<sup>2</sup> For specification of Mode 1 see System Description Compact Disc Read Only Memory, N.V. Philips and Sony Corporation, p 100-112.

VIII.3 Data Retrieval Structure

---

**VIII.3 Data Retrieval Structure**

**3.1 General**

All Base Case Systems are capable of fully processing the File structure specified in Chapter III. The only additions and restrictions to this are given below.

**3.2 ISO 9660 File Structure**

The ISO 9660 file structure is supported by the Base Case System for Mode 1 discs and Mode 2 Form 1 discs.

VIII.4 Base Case Audio

---

**VIII.4 Base Case Audio****4.1 General**

All parts specified in Chapter IV should be supported by the Base Case System. The only additions or restrictions to this are given below.

**4.2 Audio Mixing Control****4.2.1. Base Case System Audio Sources**

The Base Case System has a mixing control unit (see IV.6.3). The required tolerances for the attenuators are as follows:

Figure VIII.3 **Attenuator tolerances in the Base Case Audio Mixer**

Value Range	Tolerance
0 dB to 20 dB	+ 1 dB - 1 dB
21 dB to 127 dB	+ 1 dB - x dB x >0

**4.2.2 External Audio Source**

Audio mixing control for external audio sources is not a part of the Base Case System.

VIII.4 Base Case Audio

---

**4.3 Audio Decoder Delay**

The delay caused by the ADPCM decoder and audio processing unit (see Chapter IV.5.1 and Figure IV.17) should be less than 27 milliseconds. This is the delay time between audio data input to the ADPCM decoder and an audible output.

**4.4 Audio Output**

A Base Case System must have an analog stereo audio output.

---

## VIII.5 Base Case Video

---

### VIII.5 Base Case Video

#### 5.1 Display

The Base Case display is, at least, a 525 line (typically 60 Hz) or 625 line (typically 50 Hz) display with two fields per frame and normal 'consumer color TV' screen resolution<sup>3</sup>. If overlay of external video is required, the display must be capable of interlace. It is **recommended** to use an analog RGB input to the display.

#### 5.2 Color Resolution

The Base Case System must have an output digital to analog converter with a color resolution of at least eight bits per R, G and B color component, in order to avoid unwanted color quantization effects.

#### 5.3 Video Functions

All video functions specified in Chapter V are to be implemented in the Base Case System except for those specifically marked as **extensions** (related to High Resolution).

#### 5.4 Image Coding Resolution

Normal, Double and High Resolution are defined in the Base Case according to Figure V.42.

#### 5.5 Display Control Program

In the Base Case, Field Control Tables (see Chapter V.4.5.1) have a maximum length of 1024 instructions per field, and Line Control Tables (see Chapter V.4.5.1) have a size of 8 instructions per line.

---

<sup>3</sup> That is a screen resolution that can resolve Normal Resolution images.



### VIII.5 Base Case Video

---

#### 5.6 Mosaic Pixel Repeat Factors

In the Base Case, the horizontal mosaic pixel repeat factor is restricted to the values 2,4,8 and 16.

#### 5.7 Mattes

The Base Case has 8 matte control registers.

#### 5.8 Video Decoder Delay

The delay caused by the access controller and video decoder enabling<sup>4</sup> an image for display and the image appearing on display should not be more than one field.

#### 5.9 External Video Source

Although the function to select between a backdrop color and external video is part of the Base Case System, input and synchronization to an external video source is a manufacturer's option.

---

<sup>4</sup> Calling of a UCM display control program function which causes an image to be displayed.

## VIII.6 Program-Related Data Representations

---

### VIII.6 Program-Related Data Representations

#### 6.1 General

All parts specified in Chapter VI should be supported by the Base Case System. The only additions or restrictions to this are given below.

#### 6.2 Executable Object Code

All executable programs must be in the executable object code format specified in VI.2.1.

#### 6.3 Character Sets

All Base Case Systems must have a standard CD-I character set in ROM. This character set must conform to the specification in VI.3.1.1. The name of the default character font module must be "font8x8".

VIII.7 CD-RTOS

---

**VIII.7 CD-RTOS**

**7.1 General**

All parts specified in Chapter VII shall be supported in the Base Case. All modules specified in Chapter VII shall be contained in ROM. The only additions or restrictions to this are given below.

**7.2 Kernel**

All Base Case systems must contain the CD-RTOS Kernel specified in VII.1.

All Base Case Systems must provide system clock ticks at intervals of 1/100 second ( $\pm 0.2\%$ ).

All Base Case Systems must use the startup procedures specified in VII.1.2

### 7.3 File Managers

All Base Case systems must contain the following file managers:

- Compact Disc File Manager (CDFM),
- User Communications Manager (UCM),
- Non-Volatile RAM file manager (NRF),
- Pipe File Manager (Pipeman).

CDFM, UCM, NRF are specified in VII.2.2, VII.2.3, VII.2.4 respectively. Pipeman is specified in chapter 10 of Appendix VII.1. All functions of these file managers are **mandatory** apart from the exceptions given below.

#### 7.3.1 CDFM ~~IS~~Setstt Functions

**SS\_Eject** This function is **recommended**. If the CD drive does not support a disc eject function this call shall return an E\$UnkSvc (unknown service request) error to the application.

**SS\_Mount** The multidisc selection function of the SS\_Mount call is **recommended**.

#### 7.3.2 CDFM File Structure Support

The ISO 9660 file structure is supported by the Base Case CDFM for Mode 1 discs and Mode 2 Form 1 discs.

#### **7.4 Device Drivers**

The device drivers for the CDFM, UCM, NRF and PIPEMAN file managers specified in VII.3 are mandatory in the Base Case including the above mentioned restrictions and additions. These drivers provide the software interface between the manufacturers hardware and the low level file manager functions. Because of the differences in hardware configurations the implementation of these drivers will vary from system to system, but the functionality must remain the same. The Base Case hardware must provide functions necessary to implement these drivers.

---

**VIII.7 CD-RTOS**

---

**7.5 Memory Usage by CD-RTOS**

The memory usage of the CD-RTOS kernel is fairly well documented. However, the dynamic usage of RAM by the CD-RTOS drivers and file managers is not well documented and should be explained at greater length. This information may impact title design in many ways.

Memory allocation is divided into two categories; static allocations for the driver static storage, and dynamic allocations for temporary storage. Static storage requirements are determined by specific drivers and will vary widely. Since all static storage requests will be made before the application starts, and will fall within the 64K allotted to the system, they will not be discussed here. The dynamic memory requirements of each file manager and/or driver will be discussed here separately. Although it is impossible to predict every allocation by a driver, it is possible to describe some details that are common across most implementations.

**7.5.1 CDFM Dynamic Memory Allocations**

CDFM allocates memory dynamically for two basic purposes. One is for I/O buffering and one is for ID tables. I/O buffering covers a number of areas: buffering the path table, buffering directory sectors, and buffering I\$Read calls.

**7.5.1.1 Path Table and Directory Buffers**

When the first path to a given disc is opened (V\_PTValid is zero), a buffer is allocated to hold the Path Table and Directory Buffer for the disc. This path may be opened by an I\$ChDir or I\$Open service request. The directory buffer will be 2048 bytes. The path table is of variable size. These two buffers will be allocated with one memory request.

If the disc is changed and a path is opened to the new disc, the old buffers will be deallocated before the new buffers are allocated.

## VIII.7 CD-RTOS

---

### 7.5.1.2 I\$Read Buffering

Every time a path is opened, CDFM will allocate a 192 byte buffer for a PCB, a PCL and a CIL, used for converting I\$Read calls into SS\_Play calls for a given path. These structures will be allocated in one block.

In addition, a 2048 byte buffer is allocated for buffering reads of Mode 2 Form 1 sectors from that file. If the application always begins reading at a sector boundary and always reads blocks of 2048 bytes (or multiples thereof), the buffer is not used. However, if the application reads less than 2048 bytes at a time or issues a read that does not include all of a given sector, CDFM will read a whole sector into this internal buffer and then copy the relevant bytes into the application's buffer. Subsequent reads that can be serviced from the buffer will be handled without accessing the disc.

The implication of this is that the most efficient method of reading Form 1 sectors is to read sector blocks and de-block them within the application. The most inefficient method of reading Form 1 sectors is to issue a read that contains the end of one sector, a whole sector following and then the beginning of the next sector. CDFM will have to issue three read requests to the CD Drive. First, it will read the first sector into its internal buffer, then copy the necessary bytes into the application buffer. Secondly, it will read as many full sectors as possible into the application buffer directly. Finally, it will read the next sector into the internal buffer and copy any remaining bytes into the application buffer.

### 7.5.1.3 ID Tables

ID tables are used by CDFM for keeping track of soundmaps. At device initialization a block of 16 IDs is reserved for soundmaps. At 16 bytes per entry, this is 256 bytes. An additional 16 bytes for a header brings the total to 272 bytes (\$110).

If an application creates more than 16 soundmaps, a second table is created. This time it will have 32 entries and the first 16 are copied into the second table. After copying, the first table's memory is returned to the system. If the application manages to fill the second table, the size will be doubled to 64 entries and the active table is again copied and de-allocated. This will continue until all available memory is used. At this point the application would receive an error indicating that the ID table is full. The size of an ID table will never be decreased, even if all entries are closed.

---

**VIII.7 CD-RTOS**

---

**7.5.1.4 Soundmaps**

Memory is allocated for soundmaps in two blocks. The first allocation is for the soundmap descriptor. It is 48 bytes. The next allocation is for the soundmap data itself. The size of the allocation depends upon the number of sound groups in the soundmap. The driver will round the number of sound groups up to an integral multiple 18 and then allocate 2304 bytes per 18 sound groups.

**7.5.2 UCM****7.5.2.1 Static Storage Allocations**

UCM controls two drivers: the video driver and the pointer driver.

The number of bytes allocated by the pointer and video drivers for static storage is unpredictable and will, in any case, be allocated from the 64K allotted to the system for initialization.

**7.5.2.2 Dynamic Memory Allocations**

The only memory allocated dynamically by UCM is 48 (\$30) bytes allocated for extra path descriptor data when an I\$Open call is made. This memory is returned when the path is closed.

The pointer driver will not typically make any additional dynamic memory allocations.

The video driver allocates dynamic memory for several purposes: ID tables, drawmaps, regions, Display Control Programs, graphic fills and as stack for scan converting polygons and patterns. The memory allocated for doing graphic fills and scan converting polygons and patterns is implementation dependent and may not be used in the same manner by all systems.

**7.5.2.2.1 ID Tables**

ID tables are used for regions, drawmaps, fonts, FCTs and LCTs. The region ID table is initialized with 64 entries. All other tables have 16 entries each. Just like soundmaps, each table has a 16 byte header and when the table is filled, the number of entries is doubled, the old one is copied into the new one and the old one is then deallocated.



## VIII.7 CD-RTOS

---

### 7.5.2.2.2 Drawmaps

Memory is allocated for drawmaps in several blocks. The exact number and size of the blocks depends upon the type of drawmap allocated. The first allocation is for the drawmap descriptor. It is 256 bytes and is allocated from generic memory (color 0). After this, the driver may allocate a block for a working pattern. If the drawmap type is CLUT4, the block is 128 bytes. For CLUT7 and CLUT8, the size is 256 bytes. For RGB555, the size is 512 bytes. DYUV and RL drawmaps do not have memory allocated for a working pattern. This memory is allocated from generic memory (color 0).

The next allocation is for the primary graphic display memory (the drawmap itself). The size of the allocation depends upon the algorithm described in section 2.3.4.1 (DM\_Creat). RGB555 drawmaps will have two blocks allocated, one from each plane.

The final block is for the Line Address Table. It is allocated from generic memory and consists of one four byte entry for each line of the drawmap. These are physical lines, not logical lines, so a LAT for a normal resolution drawmap will have half as many entries as that for a high resolution drawmap. There is one extra line allocated in the LAT that the video driver uses for bookkeeping. The LAT pointer in the drawmap descriptor actually points to the second line of the LAT.

For RGB555, 2 LATs are allocated. In this case, the extra line for bookkeeping is only used in the primary LAT.

### 7.5.2.2.3 Regions

When an application creates a region, the driver allocates a block of memory for the region descriptor plus the transition data table. This is all contiguous memory. However, since it is often difficult or impossible to calculate the size of the transition table until after all the points are calculated the driver first allocates a temporary block to use and then allocates more as necessary, until all points are calculated. The size of this temporary block is 8k.

When all points are calculated the driver allocates the real region descriptor and table and then copies the data into the real table. After that, the temporary region area is de-allocated. This is an area where there is guaranteed fragmentation. Of course, if the region is rectangular, then no temporary buffer is used, since there are no transition points.

---

**VIII.7 CD-RTOS**

---

**7.5.2.2.4 FCTs and LCTs**

When creating an LCT, the driver will take the number of lines requested, add two lines, divide by 2 for normal resolution, and then multiply by 64 bytes per line.

For FCTs, the driver will allocate a block for the number of instructions specified (plus one for the link instruction) times 4 bytes per instruction. If resolution is specified as high resolution, then this number is doubled.

Additionally, when the application executes DC\_Exec, a shadow FCT is allocated and the contents of the user's FCT is copied into the shadow FCT. The shadow FCT is larger than the user FCT by 10 instructions (40 bytes) for normal resolution and 20 instructions (80 bytes) for high resolution. This is to accommodate the first line of the LCT linked to by the FCT and some extra instructions to accommodate some features of the hardware.

If there is a different FCT active at the time, the current shadow FCT is deleted after the new shadow FCT is initialized and executed. This may also cause fragmentation if an application is shifting back and forth between active FCTs very often.

**7.5.2.2.5 Fill Routines**

The driver may allocate some space when it is executing a flood fill or boundary fill command. This is used as a stack for the recursive fill routine. Again, this temporary stack is 8k. It is de-allocated when the fill is completed. This command should not cause fragmentation.

**7.5.2.2.6 Polygons**

Finally, the driver allocates some temporary stack space (32 bytes) when it scan converts a polygon. This is used when it is creating polygonal regions as well as drawing polygons in a drawmap. When drawing a polygon, an additional space is allocated for vertex array. The number of bytes allocated is the number of vertices plus one divided by four. All of this memory is deallocated after the operation is completed, so it should not cause a problem of fragmentation.

## VIII.7 CD-RTOS

---

### 7.5.3 NRF

NRF will make a dynamic memory allocation when a path is opened or a file is created. Sixty-two bytes are allocated from generic RAM (color 0) for extra path descriptor variables. This memory is deallocated when the path is closed. When a file is deleted, the 62 bytes will be allocated for the duration of the call, and then deallocated when the call is completed.

Additionally, a block will be allocated during each read or write system call. This block will be the size of the data being read or written rounded up to a 16 byte boundary. It will be deallocated when the call is completed.

### 7.5.4 Pipeman

Pipeman does not make any dynamic allocations after initialization. The only requirements for this device are 96 (\$60) bytes used for the static variable storage. When a pipe is created with a size greater than the default (96 bytes), the memory is allocated dynamically.

---

**VIII.8 System Resources and Performance**

---

**VIII.8 System Resources and Performance****8.1 General**

In order to assure the consistent functioning, from the application's point of view, of a CD-I system, a sufficient set of system resources and performance attributes need be defined. It should be, however, noted that the actual, test of the application's conformance to the Base Case Systems' capabilities can only be made by 'playing' the disc in question on a CD-I Base Case System. Moreover, the actual test of whether a system conforms to, at the least, the Base Case System capabilities can only be made by 'playing' a reference or validation CD-I disc<sup>5</sup> on the system in question. Given these two points this section of the Base Case chapter specifies the key system resources required by the Base Case System.

**8.2 MPU**

A 16-bit data bus version of the 68000 family of microprocessors (e.g. 68000, 68010 or 68070).

**8.3 RAM**

A total RAM (i.e. video + audio + system RAM) of a minimum of 1MB organized in at least two banks of 512KB each.

Although CD-RTOS will allocate memory dynamically on behalf of processes as it expands its internal tables and data structures, the operating system components (kernel, file managers and device drivers) should use a maximum of 64KB of RAM, split evenly between the banks.

---

<sup>5</sup> Reference (validation) CD-I discs are available with the required performance figures defining the minimum level of performance to be achieved by any CD-I system when "playing" this Reference CD-I disc.

For further information enquire at the address given in the Preface of this document.

## VIII.8 System Resources and Performance

---

### 8.4 ROM

As required to store the Base Case CD-RTOS, the CSD and any supporting library functions.

### 8.5 NVRAM

A non-volatile memory of at least 8KB.

### 8.6 DMA

A means to transfer data to and from the CD-control unit (see Figure VIII.2) and system memory, that can perform the data transfer independently of the MPU once it is activated (e.g. a DMA-channel or intelligent coprocessor). In addition, a means to copy contiguous areas in system memory independently of the MPU (e.g. a DMA-channel with memory-to-memory capability).

### 8.7 CD-Control Unit

A CD-control unit which is able to decode the subheader codes including:

- masking of the file number
- selecting channels and
- channel selection for direct decoding by the ADPCM decoder
- re-interleaving audio, video and program related data streams in real-time.
- capable of selective interrupt generation upon receiving any bits of the submode byte.

### 8.8 Interrupt

All devices (i.e. X-Y pointing device, CD-controller unit, DMA controller, timer, video decoder) are interrupt driven.

## VIII.8 System Resources and Performance

---

### 8.9 X-Y Device

At least one X-Y device. The X-Y device must be capable of accessing each pixel of the display at Normal Resolution and must contain at least two 'trigger' buttons.

### 8.10 Clock-Calendar

The clock calendar must:

- have a resolution of one second
- have an accuracy of better than one minute per month.
- be capable of counting:
  - seconds
  - minutes
  - hours
  - days
  - months
  - years (two digit)
- be capable of taking into account leap years.

### 8.11 CD-Player

A ready-to-use Base Case System in stand alone form can have the CD-player integrated into it or, in interfaceable form, can have the CD-player connected to it. In either case, it is **mandatory** that the access delay from the invocation of a 'Play, CD-DA, Read or Raw' command (for a given disc address) to passing the data requested (at the start address) to the CD-control unit should not be greater than 3 seconds for the worst case disc seek covering the full radius of the disc.

For seeks in the proximity of the current head position ( $\pm 20$  Mbytes) a 1 second maximum is specified. There is a linear relationship of time versus distance for intermediate positions.

Note that timing to spin-up the disc and for opening files is not included in the figure.

**Chapter IX      Full Motion Extension****A T T E N T I O N**

The inclusion of the CD-I Full Motion Extension specification (CD-I FMV) contained in this Chapter IX of the CD-I Full Functional Specification (Green Book) does not imply that existing license agreements covering CD-I Players and/or CD-I Discs are intended to include CD-I FMV applications.

The usage of the word "Extension" in the above mentioned Chapter IX is only for technical reasons, where CD-I Full Motion is an extension of the Base Case player and in no way implies or grants an extension to any Philips License covering CD-I Players or CD-I Discs.

Licenses covering Players and/or Discs for CD-I FMV applications will be granted to interested companies by means of separate license agreements.

This page is intentionally left blank



Table of Contents

---

<b>Chapter IX</b>	<b>Full Motion Extension</b>	<b>Page</b>
IX.1	Introduction	IX-1
1.1	Scope	IX-1
1.2	Full Motion Functions	IX-1
1.3	Applied Coding for Full Motion System	IX-1
1.4	Structure of the Full Motion Specification	IX-2
IX.2	Disc Format for Full Motion	IX-3
2.1	General	IX-3
2.2	Subcode P and Q channels	IX-3
2.3	CD-I Tracks with Full Motion Data	IX-3
2.4	Sectors with Full Motion Data	IX-3
2.5	CD-I Disc Encoder and Decoder Model	IX-4
IX.3	Data Retrieval Structure	IX-5
3.1	General	IX-5
3.2	Files with Full Motion data	IX-5
3.2.1	General	IX-5
3.2.2	CD-I Channels	IX-5
3.2.3	Records	IX-5
IX.4	MPEG Video	IX-7
4.1	General MPEG Video Encoding	IX-7
4.2	MPEG Video Data Structure	IX-7
4.2.1	General	IX-7
4.2.2	CD-I Files with MPEG Video Data	IX-7
4.2.3	CD-I Channels with MPEG Video Data	IX-7
4.2.4	MPEG video sectors	IX-8
4.2.4.1	General	IX-8
4.2.4.2	Subheader Bytes	IX-8
4.2.4.2.1	File Number	IX-8
4.2.4.2.2	Channel Number	IX-8
4.2.4.2.3	Submode	IX-9
4.2.4.2.4	Coding Information	IX-10
4.2.4.3	MPEG Video Sector Interleaving	IX-10

## Table of Contents

	<b>Page</b>
4.3 MPEG Parameters	IX-11
4.3.1 MPEG System Parameters	IX-11
4.3.1.1 General	IX-11
4.3.1.2 System Clock Reference	IX-12
4.3.1.3 Mux Rate	IX-13
4.3.1.4 System Header Parameters	IX-14
4.3.1.5 Stream ID	IX-15
4.3.1.6 STD Buffer Size	IX-15
4.3.1.7 Time Stamps	IX-15
4.3.1.8 Packet Rate	IX-16
4.3.2 MPEG Video Parameters	IX-17
4.3.2.1 General	IX-17
4.3.2.2 Picture Size	IX-18
4.3.2.3 Pixel Aspect Ratio	IX-18
4.3.2.4 Picture Rate	IX-19
4.3.2.5 Bitrate	IX-19
4.3.2.6 VBV Buffer Size	IX-20
4.3.2.7 Motion Vectors	IX-20
4.3.2.8 Constrained Parameter Flag	IX-21
4.3.2.9 Picture Coding Type	IX-21
4.3.2.10 User Data	IX-21
4.3.2.11 Extension Data	IX-21
4.3.3 Entrypoints	IX-22
4.3.4 Access to MPEG Video streams	IX-22
4.3.5 Length of an MPEG Video stream	IX-22
4.4 Presentation of MPEG Video	IX-23
4.4.1 Full Motion Video Plane	IX-23
4.4.2 Mixing with CD-I Video	IX-23
4.4.3 Display Control Functions	IX-24
4.4.3.1 Display Window	IX-24
4.4.3.1.1 General	IX-24
4.4.3.1.2 Size of Display Window	IX-24
4.4.3.1.3 Position within Decoded Picture	IX-25
4.4.3.1.4 Scrolling, Opening and Closing	IX-25
4.4.3.1.5 Blanking	IX-26
4.4.3.1.6 Position within Full Motion Video Plane	IX-26
4.4.3.2 Frame Rate Conversion	IX-27

## Table of Contents

	<b>Page</b>
4.4.4 Playback Control Functions	IX-28
4.4.4.1 General	IX-28
4.4.4.2 Access Address in MPEG Video stream	IX-29
4.4.4.3 Play Forward	IX-31
4.4.4.4 Slow Motion Forward	IX-31
4.4.4.5 Freeze	IX-31
4.4.4.6 Single Picture Forward	IX-32
4.4.4.6.1 General	IX-32
4.4.4.6.2 Display Period	IX-32
4.4.4.7 Scan	IX-32
4.4.4.7.1 General	IX-32
4.4.4.7.2 Display Interval	IX-33
4.4.5 MPEG Video Memory Maps	IX-34
4.5 MPEG Video Data Encoding	IX-35
4.5.1 MPEG Video Encoder Model	IX-35
4.5.2 Pre-processor	IX-35
4.5.2.1 General	IX-35
4.5.2.2 Temporal Processing	IX-36
4.5.2.3 Spatial Processing	IX-36
4.5.3 Video Encoder	IX-37
4.5.4 ISO 11172 Stream Encoder	IX-37
4.6 MPEG Video Decoding	IX-38
4.6.1 MPEG Video Decoder	IX-38
4.6.1.1 MPEG Video Decoder Model	IX-38
4.6.1.2 Transfer Buffer	IX-39
4.6.1.3 MPEG Video Decoding Unit	IX-39
4.6.1.4 MPEG Video Buffer	IX-40
4.6.2 Audio/Video Synchronization	IX-41
4.6.2.1 Synchronization Model	IX-41
4.6.2.2 Synchronization with MPEG Audio	IX-44
4.6.2.3 Synchronization with ADPCM Audio	IX-46
4.6.2.4 Synchronization with CD-DA	IX-47
4.6.2.5 Synchronization with Base Case Video	IX-47
4.6.3 Synchronization between Events	IX-48
4.6.4 Device Status Descriptor MPEG Video Decoder	IX-50
4.7 MPEG Video Data Re-arrangements	IX-51
4.8 Extended MPEG Video Playing Time	IX-51
4.9 Error Performance	IX-51

## Table of Contents

	<b>Page</b>
IX.5 MPEG Audio	IX-53
5.1 General MPEG Audio Encoding	IX-53
5.2 MPEG Audio Data Structure	IX-53
5.2.1 General	IX-53
5.2.2 Files with MPEG Audio Data	IX-53
5.2.3 Channels with MPEG Audio Data	IX-53
5.2.4 MPEG Audio Sectors	IX-54
5.2.4.1 General	IX-54
5.2.4.2 Subheader Bytes	IX-54
5.2.4.2.1 File Number	IX-54
5.2.4.2.2 Channel Number	IX-54
5.2.4.2.3 Submode	IX-55
5.2.4.2.4 Coding Information	IX-56
5.2.4.3 MPEG Audio Sector Interleaving	IX-56
5.3 MPEG Parameters	IX-57
5.3.1 MPEG System Parameters	IX-57
5.3.1.1 General	IX-57
5.3.1.2 System Clock Reference	IX-58
5.3.1.3 Mux Rate	IX-59
5.3.1.4 System Header Parameters	IX-59
5.3.1.5 Stream ID	IX-60
5.3.1.6 STD Buffer Size	IX-60
5.3.1.7 Time Stamps	IX-60
5.3.1.8 Packet Rate	IX-60
5.3.2 MPEG Audio Parameters	IX-61
5.3.2.1 General	IX-61
5.3.2.2 ID	IX-61
5.3.2.3 Layer	IX-62
5.3.2.4 Protection Bit	IX-62
5.3.2.5 Bit Rate Index	IX-63
5.3.2.6 Sampling Frequency	IX-64
5.3.2.7 Private Bit	IX-64
5.3.2.8 Mode and Mode Extension	IX-64
5.3.2.9 Copyright	IX-64
5.3.2.10 Original/home	IX-64
5.3.2.11 Emphasis	IX-64
5.3.2.12 Ancillary Data	IX-65
5.3.3 Access to MPEG Audio streams	IX-65
5.3.4 Length of an MPEG Audio stream	IX-65

Table of Contents

---

	<b>Page</b>
5.4 Presentation of MPEG Audio	IX-66
5.4.1 Mixing with CD-I Base Case Audio	IX-66
5.4.2 Mixing Control Functions	IX-66
5.4.3 Playback Control Functions	IX-67
5.4.3.1 General	IX-67
5.4.3.2 Access Address in MPEG Audio Stream	IX-68
5.4.3.3 Play Forward	IX-70
5.4.3.4 Mute	IX-70
5.4.4 MPEG Audio Memory Maps	IX-71
5.5 MPEG Audio Data Encoding	IX-72
5.5.1 MPEG Audio Encoding Model	IX-72
5.5.2 Pre-processor	IX-72
5.5.3 Audio Encoder	IX-72
5.5.4 ISO 11172 Stream Encoder	IX-73
5.6 MPEG Audio Decoding	IX-74
5.6.1 MPEG Audio Decoder	IX-74
5.6.1.1 MPEG Audio Decoding Model	IX-74
5.6.1.2 Transfer Buffer	IX-75
5.6.1.3 MPEG Audio Decoding Unit	IX-75
5.6.2 Audio/Video Synchronization	IX-76
5.6.2.1 Synchronization Model	IX-76
5.6.2.2 Synchronization with MPEG Video	IX-76
5.6.2.3 Synchronization with ADPCM Audio	IX-76
5.6.2.4 Synchronization with CD-DA	IX-76
5.6.2.5 Synchronization with Base Case Video	IX-76
5.6.3 Synchronization between Events	IX-77
5.6.4 Device Status Descriptor MPEG Audio Decoder	IX-77
5.7 MPEG Audio Data Re-arrangements	IX-78
5.8 Extended MPEG Audio Playing Time	IX-78
5.9 Error Performance	IX-78

## Table of Contents

	<b>Page</b>
IX.6 MPEG Still Picture	IX-79
6.1 General MPEG SP Encoding	IX-79
6.2 MPEG SP Data Structure	IX-79
6.2.1 General	IX-79
6.2.2 Files with MPEG SP Data	IX-79
6.2.3 Channels with MPEG SP Data	IX-80
6.2.4 MPEG SP Sectors	IX-80
6.2.4.1 General	IX-80
6.2.4.2 Subheader Bytes	IX-80
6.2.4.2.1 File Number	IX-80
6.2.4.2.2 Channel Number	IX-80
6.2.4.2.3 Submode	IX-81
6.2.4.2.4 Coding Information	IX-82
6.2.4.3 MPEG SP Sector Interleaving	IX-82
6.3 MPEG Parameters	IX-83
6.3.1 MPEG System Parameters	IX-83
6.3.2 MPEG Video Parameters	IX-84
6.3.2.1 General	IX-84
6.3.2.2 Picture Size	IX-84
6.3.2.3 Pixel Aspect Ratio	IX-84
6.3.2.4 Picture Rate	IX-85
6.3.2.5 Bitrate	IX-85
6.3.2.6 VBV Buffer Size	IX-85
6.3.2.7 Picture Coding Types	IX-86
6.3.2.8 User Data	IX-86
6.3.2.9 Extension Data	IX-86
6.3.3 Access to MPEG SP Streams	IX-86
6.3.4 Length of an MPEG SP Stream	IX-86

## Table of Contents

	<b>Page</b>
6.4 Presentation of MPEG Still Picture	IX-87
6.4.1 General	IX-87
6.4.2 Full Motion Video Plane	IX-87
6.4.3 Mixing with CD-I Video	IX-88
6.4.4 Display Control Functions	IX-88
6.4.4.1 Display Window	IX-88
6.4.4.1.1 General	IX-88
6.4.4.1.2 Size of Display Window	IX-88
6.4.4.1.3 Position within Decoded Still Picture	IX-89
6.4.4.1.4 Scrolling, Opening and Closing	IX-89
6.4.4.1.5 Blanking	IX-89
6.4.4.1.6 Position within Full Motion Video Plane	IX-90
6.4.4.2 Multiple Still Pictures	IX-90
6.4.5 Playback Control Functions	IX-91
6.4.5.1 General	IX-91
6.4.5.2 Access Address in MPEG SP stream	IX-91
6.4.5.3 Play Still Picture	IX-93
6.4.6 MPEG SP Memory Maps	IX-93
6.5 MPEG SP Data Encoding	IX-94
6.5.1 MPEG SP Encoder Model	IX-94
6.5.2 Pre-processor	IX-94
6.5.3 Video Encoder	IX-95
6.5.4 ISO 11172 Stream Encoder	IX-96
6.6 MPEG SP Decoding	IX-97
6.6.1 MPEG SP Decoder	IX-97
6.6.1.1 MPEG SP Decoder Model	IX-97
6.6.1.2 Transfer Buffer	IX-97
6.6.1.3 MPEG Video Decoding Unit	IX-97
6.6.1.4 MPEG Video Buffer	IX-98
6.6.2 Audio/Video Synchronization	IX-98
6.6.3 Synchronization between Events	IX-98
6.6.4 Device Status Descriptor MPEG SP Decoder	IX-98
6.7 MPEG SP Data Re-arrangements	IX-99
6.8 Error Performance	IX-99

## Table of Contents

	<b>Page</b>
IX.7 Memory Extension	IX-101
7.1 General	IX-101
7.2 Memory size	IX-101
7.3 Memory access	IX-101
7.4 Memory color	IX-102
IX:8 CD-RTOS for Full Motion	IX-103
8.1 General	IX-103
8.2 Full Motion Manager (moviman)	IX-104
8.2.1 Introduction	IX-104
8.2.2 Conventions in Notation	IX-104
8.2.3 MPEG Video Functions	IX-105
8.2.3.1 The Setstat and Getstat Functions	IX-105
8.2.3.1.1 MV_Abort	IX-106
8.2.3.1.2 MV_BColor	IX-107
8.2.3.1.3 MV_ChSpeed	IX-108
8.2.3.1.4 MV_Close	IX-110
8.2.3.1.5 MV_Conceal	IX-111
8.2.3.1.6 MV_Continue	IX-112
8.2.3.1.7 MV_Create	IX-114
8.2.3.1.8 MV_Freeze	IX-115
8.2.3.1.9 MV_Hide	IX-115
8.2.3.1.10 MV_ImgSize	IX-116
8.2.3.1.11 MV_Info	IX-117
8.2.3.1.12 MV_Jump	IX-118
8.2.3.1.13 MV_Loop	IX-119
8.2.3.1.14 MV_Next	IX-121
8.2.3.1.15 MV_Off	IX-123
8.2.3.1.16 MV_Org	IX-124
8.2.3.1.17 MV_Pause	IX-125
8.2.3.1.18 MV_Play	IX-126
8.2.3.1.19 MV_Pos	IX-129
8.2.3.1.20 MV_Release	IX-130
8.2.3.1.21 MV_SelStrm	IX-131
8.2.3.1.22 MV_Show	IX-132
8.2.3.1.23 MV_SLink	IX-133
8.2.3.1.24 MV_Status	IX-135
8.2.3.1.25 MV_Trigger	IX-136
8.2.3.1.26 MV_Window	IX-139
8.2.3.1.27 SS_Opt	IX-141



## Table of Contents

	<b>Page</b>
8.2.3.2 Path Handling Functions	IX-142
8.2.3.2.1 I\$Open and I\$Create	IX-142
8.2.3.2.2 I\$Close	IX-143
8.2.3.3 Data Structures	IX-144
8.2.3.3.1 MPEG Video Descriptor	IX-144
8.2.3.3.2 Speed Parameter	IX-148
8.2.3.3.3 MV_Status block	IX-149
8.2.3.3.4 Asynchronous Status Block	IX-150
8.2.3.3.5 Error Structure	IX-152
8.2.4 MPEG Audio Functions	IX-152
8.2.4.1 The Setstat and Getstat Functions	IX-152
8.2.4.1.1 MA_Abort	IX-153
8.2.4.1.2 MA_Close	IX-154
8.2.4.1.3 MA_Cntrl	IX-155
8.2.4.1.4 MA_Continue	IX-156
8.2.4.1.5 MA_Create	IX-157
8.2.4.1.6 MA_Info	IX-158
8.2.4.1.7 MA_Jump	IX-159
8.2.4.1.8 MA_Loop	IX-160
8.2.4.1.9 MA_Pause	IX-162
8.2.4.1.10 MA_Play	IX-163
8.2.4.1.11 MA_Release	IX-165
8.2.4.1.12 MA_SLink	IX-166
8.2.4.1.13 MA_Status	IX-168
8.2.4.1.14 MA_Trigger	IX-169
8.2.4.1.15 SS_Opt	IX-171
8.2.4.2 Path Handling Functions	IX-172
8.2.4.2.1 I\$Open and I\$Create	IX-172
8.2.4.2.2 I\$Close	IX-173
8.2.4.3 Data Structures	IX-174
8.2.4.3.1 MPEG Audio Descriptor	IX-174
8.2.4.3.2 MA_Status Block	IX-177
8.2.4.3.3 Asynchronous Status Block (ASY- block)	IX-180
8.2.5 MPEG Still Picture Functions	IX-181

## Table of Contents

	<b>Page</b>
8.3 Device Drivers	IX-182
8.3.1 Basic Functional Requirements of Moviman Drivers	IX-182
8.3.2 Data Structures	IX-183
8.3.2.1 Moviman Device Descriptors	IX-183
8.3.2.1.1 Standard Device Descriptor Header Fields	IX-184
8.3.2.1.2 Device Descriptor Option Fields (Initialization Table)	IX-184
8.3.2.1.3 Device Configuration Fields	IX-184
8.3.2.2 Moviman Path Descriptors	IX-185
8.3.2.2.1 Standard Path Descriptor Fields	IX-185
8.3.2.2.2 File Manager Path Descriptor Fields	IX-186
8.3.2.2.3 Option Table Path Descriptor	IX-187
8.3.2.3 Moviman Device Driver Static Storage	IX-187
8.3.2.3.1 Drive Tables	IX-188
8.3.3 Moviman Device Driver Subroutines	IX-190
8.3.3.1 The INIT Subroutine	IX-191
8.3.3.2 The READ Subroutine	IX-193
8.3.3.3 The WRITE Subroutine	IX-193
8.3.3.4 The GETSTAT and SETSTAT Subroutines	IX-193
8.3.3.4.1 General requirements for GETSTAT and SETSTAT	IX-194
8.3.3.4.2 Additional requirements for GETSTAT Subroutines	IX-194
8.3.3.4.3 Additional requirements for SETSTAT Subroutines	IX-194
8.3.3.4.3.1 MA_Waste	IX-195
8.3.3.4.3.2 MA_ReqSync	IX-196
8.3.3.4.3.3 MV_ReqSync	IX-197
8.3.3.5 The TERMINATE Subroutine	IX-198
8.3.3.6 The TRAP subroutine	IX-199
8.3.3.7 The IRQ Service Request Subroutine	IX-199

## List of Illustrations

<b>Fig. no.</b>	<b>Caption</b>	<b>Page</b>
IX.4.1	Encoding of the submode byte of an MPEG Video sector	IX-9
IX.4.2	Encoding of the coding information byte of an MPEG Video sector	IX-10
IX.4.3	Structure of ISO 11172 stream with MPEG Video data	IX-11
IX.4.4	The encoding of the Mux Rate field	IX-13
IX.4.5	Constraints for display window in next field	IX-25
IX.4.6	Example MPEG File Pointer and MPEG Video Pointer	IX-30
IX.4.7	MPEG Video Encoder Model	IX-35
IX.4.8	Example of encoding window within input picture	IX-36
IX.4.9	MPEG Video Decoder Model	IX-38
IX.4.10	Synchronization Model	IX-41
IX.4.11	MPEG Video and MPEG Audio delays from disc	IX-46
IX.4.12	MPEG Video stream with decoded and displayed pictures	IX-49
IX.4.13	MPEG Video Decoder DSD	IX-50
IX.5.1	Encoding of the submode byte of an MPEG Audio sector	IX-55
IX.5.2	Encoding of the coding information byte of an MPEG Audio sector	IX-56
IX.5.3	Available bitrate options for MPEG Audio streams	IX-63
IX.5.4	Example MPEG File Pointer and MPEG Audio Pointer	IX-69
IX.5.5	MPEG Audio Encoding Model	IX-72
IX.5.6	MPEG Audio Decoding Model	IX-74
IX.5.7	MPEG Audio Decoder DSD	IX-77
IX.6.1	Encoding of the submode byte of an MPEG SP sector	IX-81
IX.6.2	Encoding of the coding information byte of an MPEG SP	IX-82
IX.6.3	Example MPEG File Pointer and MPEG SP Pointer	IX-92
IX.6.4	MPEG SP Encoder Model	IX-94
IX.6.5	Example of encoding window within input picture	IX-95
IX.8.1	Full Motion Data Flow in CD-RTOS	IX-103
IX.8.2	The format of the color value parameter	IX-107
IX.8.3	The format of the type parameter	IX-114
IX.8.4	Signal/eventmask format	IX-137
IX.8.5	Display window	IX-139
IX.8.6	MPEG Video Descriptor	IX-144
IX.8.7	Speed Parameter	IX-148
IX.8.8	MV_Status_block	IX-149
IX.8.9	Asynchronous Status Block	IX-150
IX.8.10	Layout of the ASY_Stat field	IX-150
IX.8.11	Error Structure format	IX-152
IX.8.12	Attenuation value format	IX-155
IX.8.13	Format of the audio type parameter	IX-157

List of Illustrations

---

	<b>Page</b>
IX.8.14	Format of signal/eventmask parameter IX-170
IX.8.15	MPEG Audio Descriptor IX-174
IX.8.16	MA_Status_Block IX-177
IX.8.17	MAS_Head status word IX-179
IX.8.18	Asynchronous Status Block IX-180
IX.8.19	The layout of the ASY_Stat field IX-180
IX.8.20	Device Descriptor Option Fields IX-184
IX.8.21	Standard Path Descriptor Fields IX-185
IX.8.22	MoviMan Path Descriptor Fields IX-186
IX.8.23	Option Table of a MoviMan Path Descriptor IX-187
IX.8.24	MoviMan Device Driver Static Storage IX-188
IX.8.25	MoviMan Drive Table Format IX-189
IX.8.26	Layout of a map descriptor block IX-191

**Chapter IX Full Motion Extension****IX.1 Introduction****1.1 Scope**

The scope of this chapter is to specify the Full Motion functions, to define the format of Full Motion Data on a CD-I disc and to define the interface to applications.

**1.2 Full Motion Functions**

Full Motion extends the CD-I System with the capability to play moving natural pictures on full screen with associated audio. Another function of Full Motion is to code still pictures. Coding of natural pictures and coding of still pictures are mutually exclusive functions. For Full Motion decoding in a CD-I Player, a memory bank of at least 4Mbit is used. This memory bank can be used to extend in a CD-I System the amount of memory available to applications when the memory bank is not used for Full Motion decoding.

**1.3 Applied Coding for Full Motion System**

To code moving natural pictures, associated audio and still pictures, the Full Motion System applies the MPEG standard for coding audio-visual information, as defined in the MPEG standard ISO 11172.

In this chapter moving natural pictures and associated audio are therefore referred to as MPEG Video and MPEG Audio. A still picture is referred to as an MPEG Still Picture or an MPEG SP.

#### 1.4 Structure of the Full Motion Specification

The Full Motion specification is structured as follows. After the introduction in section 1, the general format of Full Motion data on CD-I disc is given in section 2. In section 3 the retrieval structure is specified for Full Motion data. An encoding model, the specifics of the data format on disc and a decoder model are defined for MPEG Video in section 4, for MPEG Audio in section 5 and for MPEG SP in section 6. Section 7 specifies the use of the memory bank for extending the amount of memory available to applications in a CD-I System. Based on the programming model of CD-I, the required extension of CD-RTOS for Full Motion is specified in section 8.

## IX.2 Disc Format for Full Motion

---

### IX.2 Disc Format for Full Motion

#### 2.1 General

The format of Full Motion data on CD-I disc conforms to the specification of the CD-I disc format as given in Chapter II.

#### 2.2 Subcode P and Q channels

The sub-code channels have the same technical specification for Full Motion as given in Chapter II.2.5.

#### 2.3 CD-I Tracks with Full Motion Data

In CD-I tracks with Full Motion data the same five types of CD-I sectors are distinguished as described in Chapter II, i.e. Audio, Video, Data, Empty and Message Sectors.

#### 2.4 Sectors with Full Motion Data

Three types of Full Motion data are distinguished: MPEG Video data, MPEG SP data and MPEG Audio data. MPEG Video data and MPEG SP data are stored in CD-I sectors of type Video; MPEG Audio data are stored in CD-I sectors of type Audio.

All sectors with Full Motion data are Form 2 sectors; both real-time sectors and non-real-time sectors are allowed. To indicate the type of Full Motion data in sectors, the Coding Information byte in the Subheader is used. The specification of the Coding Information byte depends on whether the data is MPEG Video, MPEG SP or MPEG Audio. The Coding Information byte is therefore specified in sections 4, 5 and 6 of this chapter.

## IX.2 Disc Format for Full Motion

---

### 2.5 CD-I Disc Encoder and Decoder Model

The CD-I encoder model and decoder model defined in Chapter II.5.1 and in Chapter II.6.1 remain applicable with the remark that the Audio Encoder and the Video Encoder are extended with capabilities to encode MPEG Video, MPEG Still Pictures and MPEG Audio. The CD-I sector data stream at the output of the CD-I sector processor may contain sectors with Full Motion data. These sectors, with MPEG Video data, MPEG SP data or MPEG Audio data are subsequently processed by means of appropriate decoders. Models of such decoders are given in sections 4, 5, and 6 of this chapter.



### IX.3 Data Retrieval Structure

---

#### IX.3 Data Retrieval Structure

##### 3.1 General

The data retrieval structure of Full Motion data conforms the structure as defined in Chapter III.

##### 3.2 Files with Full Motion data

###### 3.2.1 General

MPEG Video sectors, MPEG Audio sectors and MPEG SP sectors can be part of real-time files, as well as of non-real-time files.

###### 3.2.2 CD-I Channels

An application may define a number of CD-I channels within a file.

For reasons of compatibility with base-case players the following constraints on sectors with Full Motion data are applicable within a file. A CD-I channel with MPEG Video sectors shall neither contain sectors with base-case video data nor MPEG SP sectors. A CD-I channel with MPEG SP sectors shall neither contain sectors with base-case video data nor MPEG Video sectors. A CD-I channel with MPEG Audio sectors shall not contain ADPCM Audio sectors.

###### 3.2.3 Records

Full Motion sequences will in general consist of a series of 'sub-sequences'. In general an application may want to end a playback at the end of a 'sub-sequence'. For easy handling of 'sub-sequences' by applications, it is suggested therefore to set the End Of Record bit in sectors where a 'sub-sequence' ends.

This page is intentionally left blank

---

**IX.4 MPEG Video**

---

**IX.4 MPEG Video****4.1 General MPEG Video Encoding**

This section specifies the encoding of moving pictures by the Full Motion System. Furthermore this section defines the MPEG Video data structure, the decoding of coded MPEG Video data back to pictures, and the display of reconstructed pictures.

The Full Motion System applies the MPEG standard ISO 11172 to code moving pictures and to multiplex one or more coded video streams into one multiplexed ISO 11172 stream. In one ISO 11172 stream with MPEG Video data, one to sixteen MPEG Video streams can be multiplexed. On a disc, one or more ISO 11172 streams with MPEG Video data can be stored.

**4.2 MPEG Video Data Structure****4.2.1 General**

An MPEG Video stream is the coded representation of a sequence of moving pictures. One to sixteen MPEG Video streams can be multiplexed into one ISO 11172 stream.

**4.2.2 CD-I Files with MPEG Video Data**

A CD-I file can contain one or more ISO 11172 streams with MPEG Video data. Each ISO 11172 stream with MPEG Video data is stored in an integer number of MPEG Video sectors.

**4.2.3 CD-I Channels with MPEG Video Data**

An ISO 11172 stream with MPEG Video data can be part of only one CD-I channel. One CD-I channel may contain one or more ISO 11172 streams with MPEG Video data. However, if to be played consecutively, the physical distance between MPEG Video sectors from different ISO 11172 streams in the same CD-I channel in one file shall be at least 150 sectors, i.e. during a normal play the nominal time interval between reading MPEG Video sectors from these different ISO 11172 streams will be at least two seconds.

---

**IX.4 MPEG Video**

---

**4.2.4 MPEG video sectors****4.2.4.1 General**

Each ISO 11172 stream with MPEG Video data is stored in an integer number of MPEG Video sectors. Each MPEG Video sector cannot contain data from more than one ISO 11172 stream.

**4.2.4.2 Subheader Bytes****4.2.4.2.1 File Number**

The file number of MPEG Video sectors conforms to the definition given in Chapter II.4.5.2, Chapter III.4.4 and in Appendix II.1.

**4.2.4.2.2 Channel Number**

The channel number of MPEG Video sectors conforms to the definition from Chapter II.4.5.2, Chapter VII.2.2.3.2 (SS\_Play) and from Appendix II.2.

IX.4 MPEG Video

---

## 4.2.4.2.3 Submode

In MPEG Video sectors, the submode byte is encoded as shown in figure IX.4.1. Each MPEG Video sector is a Form 2 sector of type Video. The Form bit and the Video bit are therefore set to "1"; both the Data bit and the Audio bit are "0". The End of File bit, the Real-Time Sector bit, the Trigger bit and the End Of Record bit are coded conform Chapter II.4.5.3.

Figure IX.4.1 Encoding of the submode byte of an MPEG Video sector

Bit Number	Bit Name	Bit Value
7	End Of File (EOF)	x
6	Real-Time sector (RT)	x
5	Form (F)	1
4	Trigger (T)	x
3	Data (D)	0
2	Audio (A)	0
1	Video (V)	1
0	End Of Record (EOR)	x

- Where:
- Bit Names are defined in Chapter II.4.5.3;
  - x indicates a don't care in the definition of the submode byte of an MPEG Video sector;
  - Coding in conformance with Chapter II.4.5.3

## IX.4 MPEG Video

## 4.2.4.2.4 Coding Information

The coding information byte of an MPEG Video sector is encoded as shown in figure IX.4.2. Each MPEG Video sector is non-application specific. The ASCF bit is therefore "0". The Even/Odd Lines Flag (EOLF) bit is not needed for error concealment and is therefore set to "0". The Resolution bits indicate a SIF picture size. See video part of the MPEG standard. The Coding bits indicate MPEG coding.

Figure IX.4.2 Encoding of the coding information byte of an MPEG Video sector

Bit Number	Bit Name	Bit Value
7	ASCF	0
6	EOLF	0
5	Resolution	0
4		0
3	Coding	1
2		1
1		1
0		1

- Where:
- Bit Names are defined in Chapter V.6.3.1;
  - Resolution bits indicate SIF picture size (see MPEG standard);
  - Coding bits indicate MPEG coding.

## 4.2.4.3 MPEG Video Sector Interleaving

The MPEG Video sectors from one ISO 11172 stream are to be interleaved such, that in System Target Decoders (STD's), as defined in the MPEG standard, neither underflow nor overflow occurs.

IX.4 MPEG Video

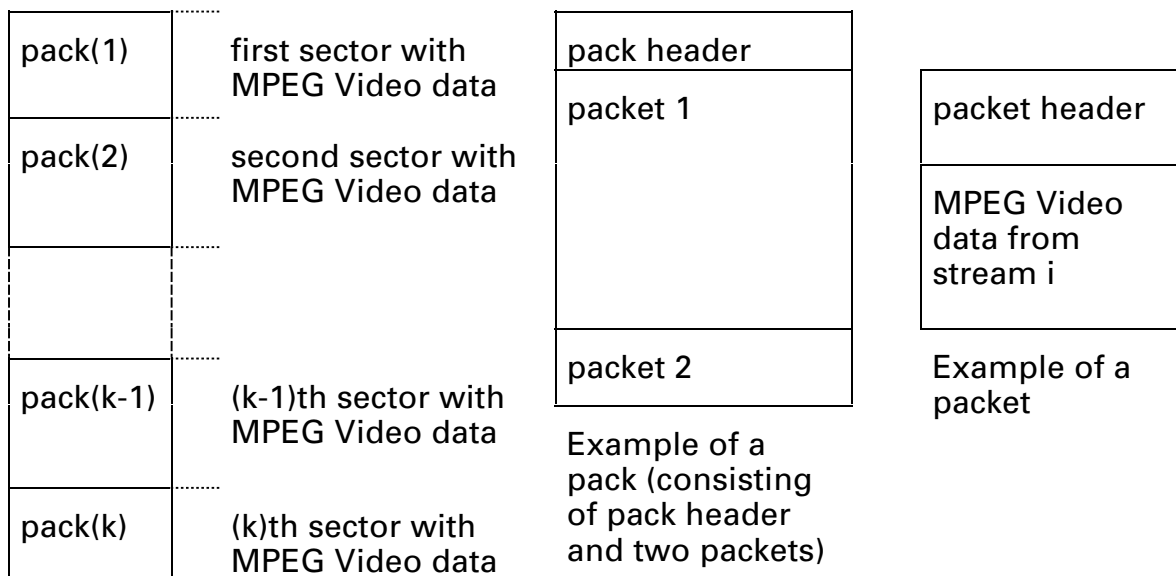
4.3 MPEG Parameters

4.3.1 MPEG System Parameters

4.3.1.1 General

To store an ISO 11172 stream with MPEG Video data in an integer number of sectors the system part of the MPEG standard ISO 11172 is applied. In each MPEG Video sector one pack is stored; for the definition of a pack see the MPEG standard. A pack consists normally of 2324 bytes, corresponding to the number of data bytes in a Video sector. The final pack of an ISO 11172 stream contains 2320 bytes, to allow for the ISO 11172 End Code of 4 bytes. Within each pack an optional system header and one or more packets are stored; for the definition of a packet see the MPEG standard. A packet can be a video packet, a padding packet, a reserved packet, a reserved data packet or a private packet. In case of a video packet, it contains MPEG video data from one video stream. Each video packet, including the packet header, consists of an even number of bytes. The first byte of a video packet must be located on a byte with an even index number within a sector (see Chapter II.4.1.2). The number of packet data bytes in each packet contained by an MPEG Video sector is even. As an example, an ISO 11172 stream with MPEG Video data is depicted in figure IX.4.3, where the ISO 11172 stream consists of k packs, stored in k sectors.

Figure IX.4.3 Structure of ISO 11172 stream with MPEG Video data



IX.4 MPEG Video

---

In ISO 11172 parameters are defined for the system part. The parameters are included in the pack header or in the packet header. Within the Full Motion Extension of CD-I some of these parameters are constrained in syntactical or semantical sense. In the following sections the constraints for MPEG Video sectors to be applied on the parameters from the system part are defined.

4.3.1.2 **System Clock Reference**

In each MPEG Video sector a pack is stored, consisting of a pack header, an optional system header and one or more packets with e.g. padding data or MPEG Video data. The System Clock Reference (SCR) parameter in the pack header indicates the intended time of arrival of the last byte of the SCR field at the input of a hypothetical System Target Decoder (STD). See the MPEG standard. The SCR is measured in periods of 90 kHz.

Within the Full Motion system the frequency of the system clock in the STD has a fixed relation to the nominal rate at which sectors are delivered from CD-I disc:

$$f_{90000}(\text{std}) = 1200 * f_{75}(\text{sector})$$

where  $f_{90000}(\text{std})$  is the nominal frequency of the system clock in the STD and  $f_{75}(\text{sector})$  is the nominal rate at which sectors are delivered from disc.

As a consequence for real-time sectors from the same ISO 11172 stream, the difference between encoded SCR values in two sectors can be calculated from the distance between both sectors:

$$\text{SCR}(\text{sector } S_2) - \text{SCR}(\text{sector } S_1) = 1200 * (D + 1)$$

where D is the number of sectors on disc between sector  $S_1$  and sector  $S_2$ .

MPEG Video streams can be played from memory with sector rates of 1, 2, 3 or 4 times the nominal sector rate. Therefore, when playing MPEG video from memory, the above formula needs modification to:

$$\text{SCR}(\text{sector } S_2) - \text{SCR}(\text{sector } S_1) = (1200/k) * N$$

where k is the speed increase factor 1, 2, 3 or 4 at which the stream is intended to be played and  
N is an integer number.



## IX.4 MPEG Video

The speed increase factor indicating at which speed an MPEG Video stream is to be played may vary in the course of an ISO 11172 stream within the Full Motion System.

At the start of an ISO 11172 stream, the system clock may have any value allowed by the range of the SCR parameter. There is no fixed relation to the system clock in other ISO 11172 streams. System clocks in STD's for different ISO 11172 streams have a fixed but arbitrary offset due to the relation of the individual system clocks to the nominal sector delivery rate.

## 4.3.1.3 Mux Rate

The Mux Rate parameter in the pack header indicates the rate at which the hypothetical System Target Decoder (STD) receives the bytes from the ISO 11172 stream in the sector. Within the Full Motion system all ISO 11172 bytes from a sector arrive at a rate defined by the nominal period of that sector. The rate is found by multiplying the total number of bytes in the sector (2352 = sync field, header field, subheader field and data field) by the sector rate. The sector rate depends on the speed at which the ISO 11172 stream is intended to be played. ISO 11172 streams can be played at 1, 2, 3 or 4 times the nominal sector rate of 75 Hz. The sector rate of an MPEG Video sector can therefore be 75, 150, 225 or 300 per second. The resulting bitrate as well as the value to be encoded in the Mux Rate field in units of 50 bytes is given in figure IX.4.4.

Figure IX.4.4      **The encoding of the Mux Rate field**

Play speed of sector	Bitrate	Mux Rate value to be encoded
1 times nominal speed	1.4112 Mb/s	3528    (= 75 * 2352 / 50)
2 times nominal speed	2.8224 Mb/s	7056    (= 150 * 2352 / 50)
3 times nominal speed	4.2336 Mb/s	10584   (= 225 * 2352 / 50)
4 times nominal speed	5.6448 Mb/s	14112   (= 300 * 2352 / 50)

## IX.4 MPEG Video

---

As a consequence of byte arrival as defined before, all ISO 11172 bytes in a sector are delivered within the nominal sector period. If for example two sectors from an ISO 11172 stream are separated by two sectors with other data, then the input to the STD consists of byte delivery during one sector period, followed by two sector periods without byte delivery to the STD and one sector period with byte delivery as during the first sector period; the resulting byte delivery is bursty.

The rate at which an ISO 11172 stream is intended to be played is defined at encoding and may vary within the same ISO 11172 stream, i.e. in each Mux Rate field either value of Figure IX.4.4 may be encoded.

### 4.3.1.4 System Header Parameters

Each ISO 11172 stream within the Full Motion system is a Constrained System Parameter Stream (CSPS stream). The CSPS Flag shall therefore be set to "1" in each System Header. In general MPEG Video sectors are interleaved with other sectors, without meeting the requirements for fixed bitrate operation. Usually the Fixed Flag is therefore set to "0". In each MPEG Video stream there is a constant relation between the video picture rate and the system clock frequency in the System Target Decoder. In each System Header the System Video Lock Flag is therefore set to "1". Although no MPEG Audio streams are included in MPEG Video sectors, also the System Audio Lock Flag is set to "1" in each System Header. The rate bound field shall not indicate a value of the mux rate parameter larger than allowed in the Full Motion system. The audio bound field shall indicate that no MPEG Audio streams are included in the multiplex. The value encoded in the STD buffer size bound field for MPEG Video streams shall not indicate a value larger than 46 KByte.

---

**IX.4 MPEG Video**

---

**4.3.1.5 Stream ID**

The Stream ID in the Packet Header specifies the type of stream stored in the packet. MPEG Video sectors do not contain MPEG Audio data. Values of the stream ID parameter indicating an MPEG audio stream are therefore not allowed. All other values of the Stream ID parameter allowed by the MPEG standard can be used. However, MPEG Video decoders within the Full Motion system will ignore packets with reserved streams as well as packets with private streams.

In case of a packet of an MPEG Video stream, the Stream ID also indicates the number of the MPEG Video stream. All Stream ID numbers allowed by the MPEG standard can be used.

**4.3.1.6 STD Buffer Size**

Each ISO 11172 stream within the Full Motion system is a CSPS stream. The picture size of each MPEG Video stream within the Full Motion system meets the requirements for the Constrained Parameters Flag in the video part of the MPEG standard. The STD Buffer Size in the Packet Header of packets with MPEG Video data shall therefore not indicate a larger value than  $46 * 1024$  Bytes. Values smaller than  $46 * 1024$  are allowed to be used.

**4.3.1.7 Time Stamps**

An MPEG Video stream is a series of one or more coded pictures. Within the Full Motion system MPEG Video streams are stored in packets. In each packet in which a coded picture commences, the Time Stamp is encoded in the packet header. A coded picture commences in a packet, if the first byte of a picture start code is present in the packet data. For Time Stamp encoding see the MPEG standard.

**4.3.1.8 Packet Rate**

The Rate at which packets shall arrive at the input of the hypothetical System Target Decoder (STD) is bounded to 300 Hz, as specified in the system part of the MPEG standard for CSPS streams. The consequences for the number of packets stored in an MPEG Video sectors depend on the speed at which the sector is intended to be played. See also Chapter IX.4.3.1.2 and Chapter IX.4.3.1.3. Next to the CSPS requirement for the packet rate, an additional requirement is to be met; applications shall not store more than four packets or system headers in one MPEG Video sector.

---

**IX.4 MPEG Video**

---

**4.3.2 MPEG Video Parameters****4.3.2.1 General**

Moving pictures are coded corresponding to the video part of the MPEG standard ISO 11172. After coding an MPEG Video stream is available. An MPEG video stream is a series of one or more MPEG video sequences. An MPEG video sequence is a series of one or more groups of pictures over which the sequence parameters (e.g. picture rate, picture size, bitrate, pixel aspect ratio) do not change. A group of pictures is a series of one or more pictures intended for random access into the sequence.

The coded representation of the next MPEG video sequence in the MPEG Video stream does not necessarily commence immediately after the coded representation of the previous video sequence. The decoding and presentation of pictures from the next MPEG video sequence does not necessarily commence immediately after the decoding and presentation of pictures from the previous MPEG video sequence.

An MPEG video stream may be a concatenation of sequences with different sequence parameters. E.g. in case of a mix of 30 Hz material from a camera and 24 Hz material from a film scanner, the picture rate will change.

In the video part of ISO 11172 the MPEG Video parameters are defined. Also "Constrained Parameter Streams" have been defined. An MPEG Video stream is a "Constrained Parameter Stream" if the stream parameters are within certain bounds. Within the Full Motion system the same bounds are applied for MPEG Video streams, with the exception that more flexibility with respect to the bitrate is allowed in the Full Motion system. The Full Motion System applies some more constraints in syntactical or semantical sense. The following sections specify all constraints from the Full Motion system on parameters of MPEG Video streams. On parameters not referred to, no specific constraints apply.

## IX.4 MPEG Video

---

### 4.3.2.2 Picture Size

By means of the MPEG video standard rectangular pictures can be coded. Within an MPEG Video sequence the picture size is constant. An MPEG Video stream may be a concatenation of MPEG Video sequences with different picture sizes. The picture size is defined at encoding. The total picture area which can be coded is limited, but can be used in a flexible way. For example wide and low pictures can be coded as well as high and narrow pictures.

The minimum size of a coded picture is one pixel by one line. The maximum picture area which can be coded in case of 30 Hz pictures is 352 pixels by 240 lines, and in case of 25 Hz pictures 352 pixels by 288 lines.

On the picture size within the Full Motion system the same constraints are applied as for "Constrained Parameter Streams" (see video part of ISO 11172). The horizontal size shall be less than or equal to 768 pixels. The vertical size shall be less than or equal to 576 lines. The total picture area shall be less than or equal to 396 macroblocks. A macroblock is a rectangular block of 16\*16 luminance pixels and corresponding chrominance pixels; see video part of ISO 11172. The picture size is bounded further by the constraint on the pixel rate: the total number of macroblocks per second shall be less than or equal to 396\*25.

### 4.3.2.3 Pixel Aspect Ratio

Within the Full Motion system no constraints on the pixel aspect ratio are applied. Applications should take into account however that Full Motion decoders do not compensate for aspect ratio distortion. It is recommended therefore to select one of the following aspect ratio options:

- 1.19 if no aspect ratio distortion is to occur in decoders producing a 525 lines output; in this case however on decoders producing a 625 lines output the aspect ratio will be distorted by approximately 13 %.
- 1.05 if no aspect ratio distortion is to occur in decoders producing a 625 lines output; in this case however on decoders producing a 525 lines output the aspect ratio will be distorted by approximately 12 %.
- 1.12 if aspect ratio distortions are equalized over decoders producing 525 lines and 625 lines outputs. In both decoders the distortion will be approximately 6 %.

The above aspect ratio values are recommended for practical use, but they cannot be coded exactly in the MPEG Video stream. The nearest value in the aspect ratio table from the MPEG standard is to be used instead.

---

**IX.4 MPEG Video**

---

**4.3.2.4 Picture Rate**

The picture rate indicates the number of pictures coded per second. Picture rates which can be used in the Full Motion system are 23.976 Hz, 24 Hz, 25 Hz, 29.97 Hz and 30 Hz, the same as allowed for "Constrained Parameter Streams".

**4.3.2.5 Bitrate**

Within the Full Motion system the Bitrate of MPEG Video is bounded to 5,000,000 bits per second. When playing an MPEG Video stream from disc, the maximum bitrate of an MPEG Video stream is approximately 1.4 Mbit/sec. MPEG Video streams can be played from memory with bitrates up to 5 Mbit/sec. Next to fixed bitrate operation, variable bitrate operation can also be applied.

An MPEG Video stream may consist of one picture. The maximum number of bits to code a single picture is 40 KByte, i.e. 327,680 bits.

---

**IX.4 MPEG Video**

---

**4.3.2.6 VBV Buffer Size**

In the MPEG standard the Video Buffering Verifier (VBV) is defined. The VBV is a hypothetical decoder with an input buffer of which the size is specified in each MPEG Video stream. As for "Constrained Parameter Streams", the maximum size of the VBV Buffer within the Full Motion System shall be 40 KByte, i.e. 40,960 Bytes, for each MPEG Video stream.

The flexibility in positioning real-time MPEG Video sectors on a disc depends on the difference between the STD Buffer Size (see Chapter IX.4.3.1.6) and the VBV Buffer Size. In general more flexibility is required for lower Bitrates of the MPEG Video stream. In case of real-time MPEG Video sectors, applications are therefore recommended to apply a VBV Buffer Size proportional to the Bitrate of the MPEG Video stream. The following formula is recommended:

$$B_{vbv} = 40 * (R / R_{ref}) \text{ KByte,}$$

where  $B_{vbv}$  is the applied size of the VBV Buffer for MPEG Video streams in real-time MPEG Video sectors;

$R$  is the Bitrate in Mbit/sec of the MPEG Video stream in a real-time MPEG Video sector.

$R_{ref}$  is a reference Bitrate; at this value of the Bitrate  $R$  the maximum VBV Buffer size is applied; the reference bitrate is defined by the application; examples are 1.856 Mbit/sec or 1.4 Mbit/sec.

**4.3.2.7 Motion Vectors**

For the range of Motion Vectors the same bound is applied as for "Constrained Parameter Streams" ('vector scale code'  $\leq 4$ , see video part of ISO 11172). As a consequence, the range of half pixel motion vectors is limited to approximately  $\pm 64$  pixels.



---

**IX.4 MPEG Video**

---

**4.3.2.8 Constrained Parameter Flag**

In the video part of ISO 11172 "Constrained Parameter Streams" have been defined. Such streams can be identified by means of a Constrained Parameter Flag. Within the Full Motion System the same constraints are applied for MPEG Video streams, with the exception that the Full Motion System allows a higher Bitrate. See Chapter IX.4.3.2.5.

In MPEG Video streams with a fixed bitrate of less than or equal to 1.856 Mbit/sec., the Constrained Parameter Flag shall be set to '1'. In MPEG Video streams with a fixed bitrate higher than 1.856 Mbit/sec., the Constrained Parameter Flag shall be set to '0'. Also in MPEG Video streams with a variable bitrate the Constrained Parameter Flag is to be set to '0'.

**4.3.2.9 Picture Coding Type**

The Picture Coding Type identifies whether a picture is an intra-coded picture (I type), predictive-coded picture (P type), bidirectionally-predictive-coded picture (B type), or intra-coded with only DC coefficients (D type). Three picture types, I type, P type and B type, are allowed within the Full Motion System; D type pictures shall not be used.

**4.3.2.10 User Data**

User data is application specific data. MPEG Video streams within the Full Motion system may contain User Data, but Full Motion Video decoders will not interpret such data.

**4.3.2.11 Extension Data**

Extension Data may be defined by ISO in future. MPEG Video streams may contain Extension Data, but the Full Motion Decoder will ignore such data.

## IX.4 MPEG Video

---

### 4.3.3 Entrypoints

An MPEG Video stream is a series of one or more video sequences. Each video sequence is a series of one or more group of pictures, starting with a sequence header. In an MPEG video stream, the sequence header with the sequence parameters may be repeated at the start of a new group of pictures. Each first picture after a sequence header is an entrypoint picture. An MPEG Video decoder can start to decode an MPEG Video stream at an entrypoint picture. To ensure that the correct parameters are used, the sequence header is to be read first.

The minimum distance between entrypoint pictures is one picture; the maximum distance is not defined, but the distance is recommended to be less than 256 pictures, about 10 seconds. For each MPEG video stream, the application specifies at encoding which pictures are entrypoint pictures. The application may e.g. define them on a regular distance or at scene changes.

### 4.3.4 Access to MPEG Video streams

Applications have random access to MPEG Video streams at entrypoint pictures. Applications can also start decoding of an MPEG Video stream at the first picture of a group of pictures or at an intra-coded picture within a group of pictures without reading a sequence header first; each intra-coded picture is an access picture. In all cases, the application is responsible for the availability of all parameters at the MPEG Video decoder. Note that P-type and B-type pictures are not access pictures.

The application is able to store sequence parameters at the decoder before start of decoding. During and after decoding, the application can read sequence parameters from the decoder. The parameters which can be read and stored are the horizontal and vertical size and the picture rate. The time code and the temporal reference can only be read. Applications should note that an invalid time code is read after an entry at an intra-coded picture within a group of pictures, until the correct time code value is decoded from the header of the next group of pictures.

### 4.3.5 Length of an MPEG Video stream

The minimum length of an MPEG Video stream is one picture. The maximum length is only limited by the available disc capacity.

---

**IX.4 MPEG Video**

---

**4.4 Presentation of MPEG Video****4.4.1 Full Motion Video Plane**

The MPEG Video decoder reconstructs pictures from the MPEG Video stream. Reconstructed pictures or parts thereof are displayed in the full motion video plane. The full motion video plane is available at the output of the decoder on RGB level in normal horizontal resolution (384 or 360 pixels on a line). The reconstructed picture or the part thereof which is displayed may be smaller than the spatial size of the full motion video plane. In areas of the full motion video plane which are not covered by the displayed part of the reconstructed picture, a background color is displayed. The application can program one background color for the full motion video plane. For each R, G and B component the integer values 0 to 255 can be programmed; to applications it is recommended however to use only values between 16 and 235; black level is at 16 and nominal white peak level is at 235.

The full motion video plane is fully synchronized with the other CD-I planes. The width and the height of the full motion video plane correspond to the width and height of the other CD-I planes in terms of normal resolution. The width of all planes is 384 or 360 pixels of normal horizontal resolution and the height of all planes is 280 or 240 lines. See also Appendix V.2.2.1. The field rate at the full motion video plane can be 50 Hz or 60 Hz, both interlaced and non-interlaced, following exactly the vertical synchronization of the other CD-I planes. The MPEG Video decoder is informed by the application on the required width of the full motion video plane.

The full motion video plane is available as External Video in the Video Decoder model. See Chapter V.2.6, Chapter V.4.1 and figure V.23. The full motion video plane replaces the backdrop plane, if the external video is enabled by setting the EV bit (External Video Enable) in the Select Image Coding Methods Instruction. See Chapter V.4.6.1 and figure V.47.

**4.4.2 Mixing with CD-I Video**

In the displayed image at the output of the Video Decoder Model of figure V.23 in Chapter V.4.1 the full motion plane will be visible in picture areas where all enabled CD-I video planes and the cursor are transparent. To allow for analog switching, the horizontal margin between the full motion video plane and the signal indicating transparency is between -1 and +1 pixel of normal horizontal resolution. This margin should be taken into account by applications.

---

**IX.4 MPEG Video**

---

**4.4.3 Display Control Functions****4.4.3.1 Display Window****4.4.3.1.1 General**

After decoding the MPEG Video stream, the reconstructed pictures are available at the Full Motion Video decoder. The application can control which part of the reconstructed picture is to be displayed. The application selects the part of the decoded picture to be displayed by means of a display window. The application also controls the position of the display window within the full motion video plane.

**4.4.3.1.2 Size of Display Window**

The display window has a rectangular shape, and cannot be larger than the decoded picture. As a consequence, the width and height of the window should be smaller than or equal to the width and height of the decoded picture. Within these conditions, the width of the window is an integer number of pixels and the height is an integer number of lines. The minimum width is zero pixels and the minimum height is zero lines.

The window should not exceed the border of the decoded picture. If the position of the window does not allow the window to remain within the picture, the MPEG Video Decoder clips the display window at the edge of the decoded picture. The application is informed about such clipping. The MPEG Video Decoder will use the clipped values of width and height of the display window until the width and height values are updated by the application.

The size of the display window can be changed every coded picture period. During play forward, the size is modified just before the display commences of the next coded picture.

IX.4 MPEG Video

---

**4.4.3.1.3 Position within Decoded Picture**

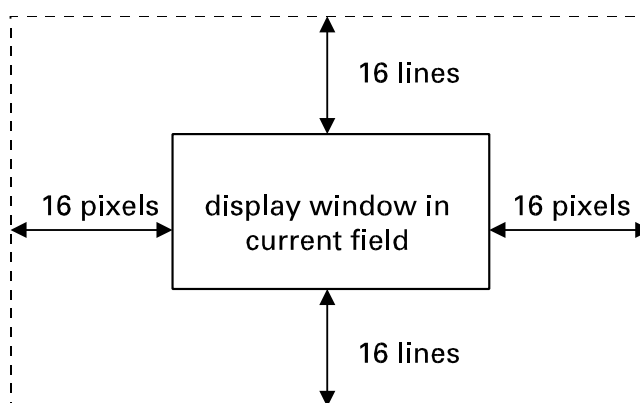
The display window can be positioned within the decoded picture by defining the address of the left upper corner of the window. This address can be any pixel/line position within the decoded picture.

Each coded picture period, the display window can be moved to an arbitrary position within the decoded picture. During play forward, the position is modified just before the display commences of the next coded picture.

**4.4.3.1.4 Scrolling, Opening and Closing**

For scrolling the position of the display window within the decoded picture can be modified each display field period. To open and close the window slowly, the size of the window can be modified each display field period. The total modification per display field period of size and position within the decoded picture is constrained. The display window which is to be displayed during the next field shall be within an area of +16 and -16 pixels or lines around the display window in the current field. See figure IX.4.5.

Figure IX.4.5 **Constraints for display window in next field**



display window in next field is to be within the dotted lines.

---

## IX.4 MPEG Video

---

### 4.4.3.1.5 Blanking

By the application, the display window can be blanked. In that case the contents of the display window will become 'black': R,G,B= 16,16,16. The blanking of the display window will remain active until the blanking is switched off by the application. After removing the blanking, the selected part of the decoded picture will be available again in the display window. In case no video sequence is played back, the blanking is switched off at the first vertical sync after issuing the command. When a video sequence is played back, the blanking is switched off at the vertical sync immediately prior to the presentation of the next decoded picture after issuing the command.

### 4.4.3.1.6 Position within Full Motion Video Plane

Within the full motion video plane one display window can be displayed simultaneously. The position of the display window within the full motion video plane can be controlled by the application by defining the position of the left upper corner of the window within the plane. The window can be positioned with an horizontal accuracy of one pixel of normal horizontal resolution and a vertical accuracy of one line.

Each coded picture period, the display window can be moved to an arbitrary position within the full motion video plane. The MPEG Video Decoder accepts a modification of the position at the start of a new coded picture period.

In order to make scrolling possible, the position of the display window within the full motion video plane can also be modified each display period. The displacement per display period is constrained: the displacement is to be in the range between +16 and -16 pixels or lines.

If the window is positioned such that the window exceeds the border of the plane, the part of the window which exceeds the border of the plane is not displayed.

#### 4.4.3.2 Frame Rate Conversion

After decoding reconstructed pictures are available at the picture rate at which they are coded. The pictures are to be displayed in the full motion plane at a display rate of 50 Hz or 60 Hz. A frame rate conversion is therefore to be applied by the MPEG Video decoder from the coded picture rate to the required display rate. Conversion is supported from each coded picture rate allowed in the Full Motion system. The conversion is only in temporal direction, without any effect on picture size in terms of pixels and lines. As a consequence the aspect ratio of the picture may be effected. The algorithm to be applied for the frame rate conversion is at the discretion of the decoder.

---

**IX.4 MPEG Video**

---

**4.4.4 Playback Control Functions****4.4.4.1 General**

An MPEG Video Decoder can playback an MPEG Video stream multiplexed into an ISO 11172 stream. Simultaneously an MPEG Video Decoder can play only one MPEG Video stream from one ISO 11172 stream. During a playback mode an application shall not switch to another ISO 11172 stream. Switching to another ISO 11172 stream is only possible by first aborting the playback mode of the current stream.

When during a playback mode the played MPEG Video stream or the ISO 11172 stream ends, i.e. a Sequence End Code or the ISO 11172 End Code is encountered in the stream, the MPEG Video playback mode remains active. Upon request of the application, the MPEG Video Decoder informs the application during each playback mode on the occurrence of these MPEG Video events:

- a Sequence End Code (see video part of MPEG standard) is encountered in the played MPEG Video stream at the input of the MPEG Video Decoder;
- an ISO 11172 End Code (see system part of MPEG standard) is encountered in the ISO 11172 stream at the input of the MPEG Video Decoder.

Based on above events, the application can control the MPEG Video Decoder.

When no further decoded pictures from an MPEG Video stream are to be displayed, the MPEG Video Decoder shall continue to display the last correctly decoded picture in the display window. The application proceeds to control the display window and defines whether this picture is to be displayed or not; see Chapter IX.4.4.3.1.7. The picture will be displayed until the first decoded picture from further MPEG Video data is to be displayed.

The application can switch between different MPEG Video streams within an ISO 11172 stream. After switching to a new stream, the MPEG Video decoder will not accept data from the previous stream anymore. The decoding of the new stream has the highest priority; the MPEG Video decoder may decide not to finalize completely the decoding of the previous stream. Applications are recommended before switching to another stream to first freeze the decoded picture of the current stream. The first picture decoded from the new MPEG Video stream is the first entry point picture of which the first byte of the picture start code is encountered in that stream. The application decides whether the last decoded picture from the previous stream is displayed (as a still picture) until decoded pictures from the next stream start being displayed.



## IX.4 MPEG Video

---

Five playback modes are distinguished in the MPEG video decoder in the Full Motion System: play forward, freeze, single picture forward, slow motion forward and scan. The functionality of these five modes is described in the following sections. The functionality is provided for playback from disc as well as from memory, unless stated otherwise.

### 4.4.4.2 Access Address in MPEG Video stream

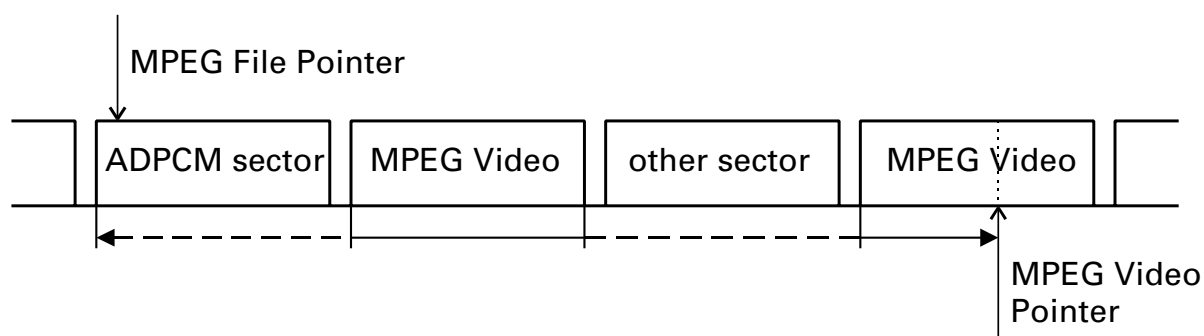
Each playback mode can start at entrypoint pictures or other access pictures; see Chapter IX.4.3.3 and Chapter IX.4.3.4. In case of a play from disc, the application can address an access position by means of an MPEG File Pointer and an MPEG Video Pointer. For byte indices in MPEG Video sectors see e.g. Chapter II.4.8.1.

The MPEG File Pointer addresses the first sector to be retrieved from disc. This sector may be related to playback requirements of e.g. MPEG Audio or ADPCM. The sector addressed by the MPEG File Pointer may therefore be any sector from the file with the MPEG Video stream to be played. Applications should take into account that file pointers use a sector size of 2048 bytes to calculate a position within a file; see Chapter VII.2.2.3.2, the description of a function to issue a seek command to the disc drive.

The MPEG Video Pointer addresses the first byte to be accessed in the ISO 11172 stream. The MPEG Video Pointer shall address the first byte of a pack start code, or the first byte of a sequence header code, group start code or picture start code. See system part and video part of ISO 11172.

In case of an access from disc, the MPEG Video Pointer addresses the first byte of the access by indicating the index of that byte within the ISO 11172 stream from the selected CD-I channel. Note that one CD-I channel may contain sub-subsequent ISO 11172 streams with MPEG Video data. See Chapter IX.4.2.3. The first byte of the ISO 11172 stream retrieved from disc in the selected CD-I channel has index zero. See also the example in figure IX.4.6.

Figure IX.4.6 Example MPEG File Pointer and MPEG Video Pointer



MPEG File Pointer points to ADPCM sector. Data retrieval from disc will therefore start with this ADPCM sector.

MPEG Video Pointer indicates the index of a byte in the ISO 11172 stream from the selected CD-I channel. If this Pointer has the value zero, the access will start at the pack header from the first MPEG Video sector retrieved from disc in the selected CD-I channel.

In this example the MPEG Video Pointer indicates the first byte of (e.g) a sequence header code. In the example the MPEG Video Pointer equals 3548; therefore the first accessed byte of the MPEG Video stream is the 1225th byte ( $3548 - 2324 + 1 = 1225$ ) of the second MPEG Video sector from the selected CD-I channel retrieved from disc.

In case of an access from memory, only the MPEG Video Pointer is used. In such a case, the MPEG Video Pointer indicates the index of a byte from the ISO 11172 stream stored in memory. The first byte of such an ISO 11172 stream has index zero.

Decoding will start at the first access picture encountered in the MPEG Video stream from the byte referred to by the MPEG Video Pointer. A picture is encountered in the stream if the first byte of the picture start code is encountered. Parameters from a sequence header or from the group of pictures layer are read by the decoder before starting the decoding, if the first byte of a sequence header code or of a group start code from the accessed MPEG Video stream is read prior to the first byte of the picture start code of the access picture.

---

**IX.4 MPEG Video**

---

**4.4.4.3 Play Forward**

In the play forward mode an MPEG Video stream is decoded and played back in normal speed forward. A play forward can be paused at each picture and can be continued at the paused picture. A play forward can start at an entrypoint picture or another access picture; see IX.4.3.3 and IX.4.3.4. The play forward can be started directly from both the single picture forward mode and the slow motion mode at the last picture decoded in the previous mode. From the freeze mode and the scan mode, a play forward can be started only at the first access picture encountered in the MPEG Video stream.

**4.4.4.4 Slow Motion Forward**

In the slow motion forward mode an MPEG Video stream is decoded and played back in slow motion in forward direction. During the slow motion mode seven speeds are available, defined by the formula  $(1/m) * \text{normal speed}$ , where  $m = 2, 3, 4, \dots, 8$ . At each coded picture period the application can switch from one speed to another. The slow motion mode can be started, paused and continued on the same conditions as the play forward mode; see Chapter IX.4.4.4.3. Transitions between the play forward mode and the slow motion forward mode are possible at each picture.

**4.4.4.5 Freeze**

The freeze mode can be entered from the play forward mode. At entering the freeze mode, the current picture is frozen and the decoding stops, but the MPEG Video stream continues. From the freeze mode the mode from which the freeze mode was entered can be continued. In that case, decoding will start at the next access picture encountered in the MPEG video stream. The freeze mode can be used both in case of playback from disc and in case of playback from memory.

---

**IX.4 MPEG Video**

---

**4.4.4.6 Single Picture Forward****4.4.4.6.1 General**

In the single picture forward mode pictures from the MPEG Video stream are decoded and displayed one by one. The single picture forward mode can be entered directly from the play forward and the slow motion forward mode at each picture. The single picture forward mode can be started also from entrypoint pictures or other access pictures; see IX.4.3.3 and IX.4.3.4.

**4.4.4.6.2 Display Period**

The application can control the length of the period during which one picture is displayed, by controlling the time interval between successive single picture forward commands. The minimum length of the display period of one picture is the coded picture period. The maximum length is not constrained.

**4.4.4.7 Scan****4.4.4.7.1 General**

During the scan modes entrypoint pictures or other access pictures (see Chapters IX.4.3.3 and IX.4.3.4) are displayed as still pictures. Each scanned picture is displayed during a time interval, of which the length can be controlled by applications. Which pictures are scanned is defined in applications e.g. from a scan table. The scan may be in forward, reverse or random direction. The scanned pictures may be from different sequences from different MPEG video streams. The application is responsible for the availability of all required decoding parameters at the decoder.

Each picture to be scanned is searched in the MPEG Video stream, decoded and finally displayed during a time interval, of which the nominal length is specified by the application. During this interval the application can search for the next picture in the MPEG Video stream.

**4.4.4.7.2 Display Interval**

In the scan mode, the next picture to be scanned is searched and decoded during the display interval of the previously scanned picture. The nominal length of this interval is specified by the application. However, depending on the response time of the CD-I player system, the MPEG Video Decoder may need to extend the nominal display interval, to wait for the availability of the next picture. By trial and error, applications may adjust the length of the time interval to the response time of the CD-I player system.

The application can specify the nominal display interval in units of 10 msec. The nominal display interval can be programmed as a value of  $(p * 10)$  msec, where  $p$  is an 31 bit integer value.

---

**IX.4 MPEG Video**

---

**4.4.5 MPEG Video Memory Maps**

The Full Motion System supports playback of an ISO 11172 stream, or a continuous part thereof, from memory. An MPEG Video stream multiplexed into an ISO 11172 stream, or a part thereof, shall be played from memory only, if the ISO 11172 stream meets the specifications from the ISO 11172 standard, as well as the constraints for MPEG parameters as specified in Chapter IX.4.3. The map in memory of an ISO 11172 stream, played from memory, shall commence with the first byte of a pack start code.

Play back of MPEG Video from disc and play from memory are mutually exclusive functions. During an MPEG Video play from memory, the bitrate for ISO 11172 streams and for MPEG Video streams is bounded by the constraints specified in Chapters IX.4.3.1.3 and IX.4.3.2.5.

A play of an MPEG Video map starts with the byte in the ISO 11172 stream indicated by the MPEG Video Pointer. The MPEG Video Pointer indicates the first byte of a pack start code or the first byte of a sequence header code, group start code or picture start code. See Chapter IX.4.4.4.2. The MPEG Video stream can be played from the first access picture in the stream (see Chapter IX.4.4.4.2) until the last picture included in the stream.

With memory maps, MPEG Video loops can be created, by means of which MPEG Video sequences are repeated. The number of loops to be played is specified by the application. The first and last picture in the loop are defined by the application by means of two MPEG Video Pointers. The MPEG Video Pointer defining the first picture in the loop indicates the first byte of a pack start code or the first byte of a sequence header code, group start code or picture start code. See Chapter IX.4.4.4.2. The MPEG Video Pointer defining the last picture in the loop indicates the byte following the last byte of the coded representation of the last picture in the loop.

An MPEG Video loop ends when all pictures are decoded from the selected MPEG Video stream between both MPEG Video Pointers to the ISO 11172 stream. At the start of the next loop, there will be some delay before the first picture from the next loop is displayed. The length of this delay depends on the timing characteristics of the stream, especially on the encoded values in the first SCR field and the first Time Stamp field in the loop. See the MPEG standard; see also Chapter IX.4.3.1.2 and Chapter IX.4.3.1.7. During this delay, the MPEG Video Decoder displays the last decoded picture from the previous loop.

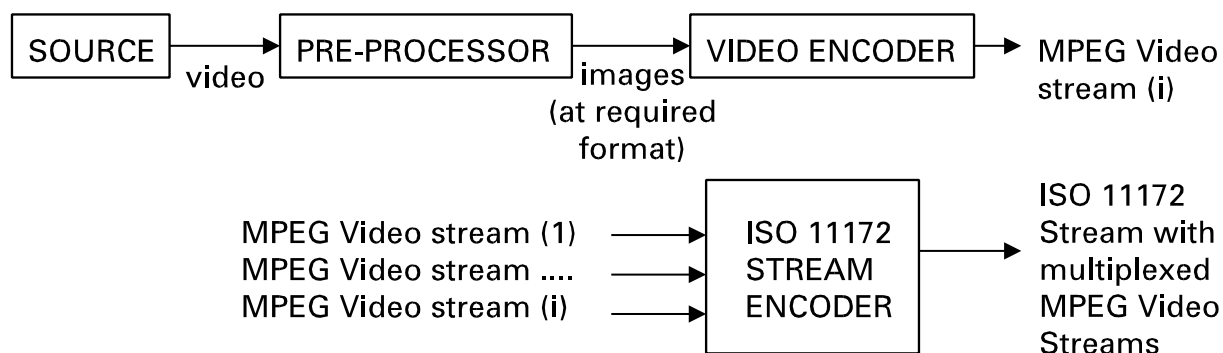
## IX.4 MPEG Video

## 4.5 MPEG Video Data Encoding

## 4.5.1 MPEG Video Encoder Model

The MPEG Video Encoder Model is depicted in figure IX.4.7. The source, e.g. a video tape recorder, delivers the video signal to the pre-processor; output of the pre-processor are the images which are to be coded. Pre-processed images have the format which is used for encoding. The images are input to the video encoder; the video encoder includes motion estimation. At the output of the video encoder, a coded MPEG Video stream with the required bitrate is available. Finally one or more MPEG Video streams are input to the ISO 11172 stream encoder, which produces an ISO 11172 stream at the format required for storage in the datafield of an MPEG Video sector.

Figure IX.4.7 MPEG Video Encoder Model



## 4.5.2 Pre-processor

## 4.5.2.1 General

MPEG Video encoding requires an input format, which is different from the usual video format from e.g. a video tape recorder or a film scanner. On the video output from the source therefore pre-processing is applied. In case of an analog input signal, A to D conversion is included in the pre-processing, as encoding requires a digital input format.

At the output of the pre-processor images are made available with the picture size and the picture rate at which they are encoded. To obtain the required spatial format and picture rate, processing in the temporal and the spatial domain is needed. Although the temporal and spatial processing may be non-separable processes, they are described separately in the following two sub-sections.

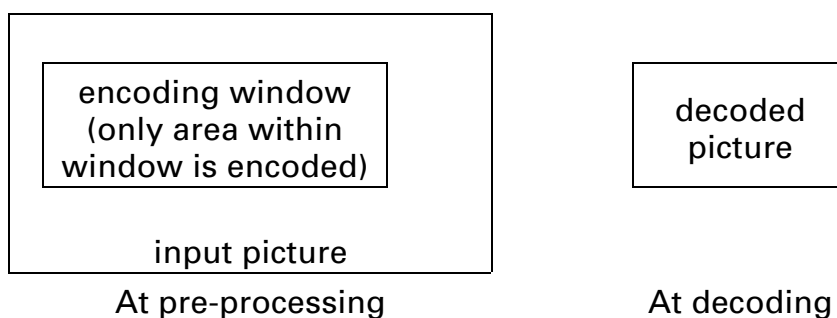
### 4.5.2.2 Temporal Processing

Temporal processing is required to convert the 60 or 50 Hz frame rate of the input signal to the picture rate used at encoding. The processing may be different for source material originating from film and from video camera. In case of film, a suitable algorithm can be used to recover the film images from the video data. In general, 50 Hz source material will be coded on 25 Hz, while 60 Hz material is expected to be coded at 30 (or 29.97) Hz in case of input originating from a video camera and at 24 (or 23.976) Hz in case of input originating from a film scanner. The techniques used for the temporal processing are at the discretion of the implementation.

### 4.5.2.3 Spatial Processing

Two types of spatial processing may be required: filtering and windowing. Filtering may be needed to reduce the spatial resolution, e.g. before sub-sampling. Which filters are applied is at the discretion of the implementation. If only a part of the input picture is to be coded, the area which is to be coded can be defined by means of an encoding window. The size of the window is constant for each MPEG Video stream. See also Chapter IX.4.3.2.2. The size of the window is limited by the constraints for the size of the coded picture. The encoding window can be positioned within input pictures with several spatial resolutions. For an example see figure IX.4.8.

Figure IX.4.8 **Example of encoding window within input picture**



The area within the encoding window will be coded; at decoding the part of the input picture which is coded will be reconstructed.



## IX.4 MPEG Video

---

### 4.5.3 Video Encoder

The video encoder produces an MPEG Video stream from the input pictures provided at the input. Within the flexibility provided by ISO 11172 and within the constraints described in Chapter IX.4.3.2, it is at the discretion of the implementation which encoding techniques are applied.

Parameters used during encoding depend on the application. Entrypoint or other access pictures are to be defined, as well as slices within pictures (see video part of ISO 11172). In the case of constant bitrate operation, the bitrate is to be specified before encoding.

### 4.5.4 ISO 11172 Stream Encoder

The ISO 11172 stream encoder multiplexes one or more MPEG Video streams into one ISO 11172 stream. The format of the produced ISO 11172 stream meets the constraints described in Chapter IX.4.3.1. The multiplex depends on which sectors are allocated as MPEG Video sectors. The MPEG Video sector allocation is therefore provided before multiplexing or is specified by the ISO 11172 stream encoder. The applied multiplex technique is at the discretion of the implementation.

## IX.4 MPEG Video

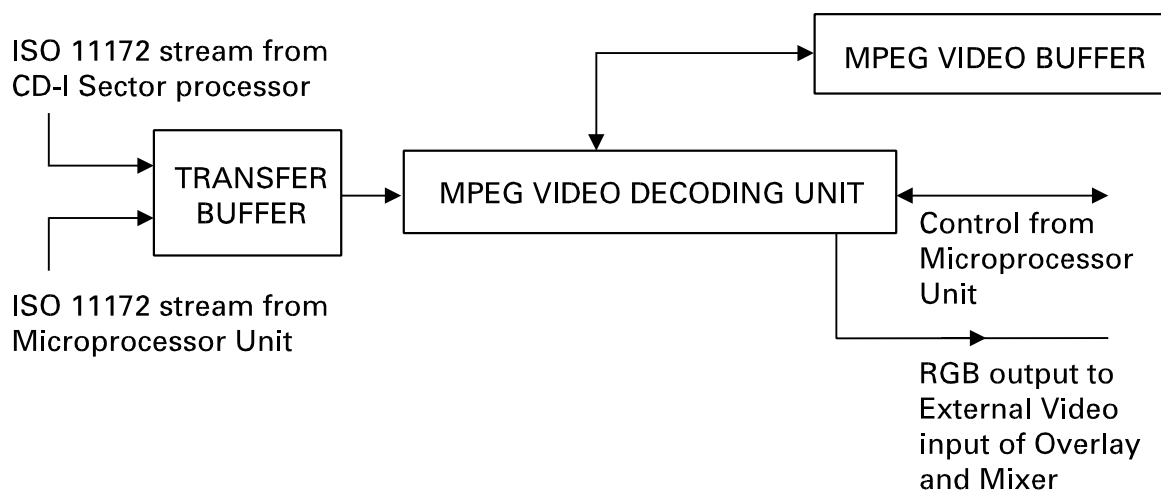
## 4.6 MPEG Video Decoding

## 4.6.1 MPEG Video Decoder

## 4.6.1.1 MPEG Video Decoder Model

The MPEG Video decoder model is given in figure IX.4.9. The CD-I Sector Processor (see Chapter II.6.1) or the Microprocessor Unit is delivering an ISO 11172 stream from MPEG Video sectors in the selected CD-I channel. The ISO 11172 stream is stored in a Transfer Buffer. Subsequently, the MPEG Video Decoding Unit reads the ISO 11172 stream from the Transfer Buffer to decode one of the MPEG Video streams in the ISO 11172 stream. Which MPEG Video stream is decoded is controlled from the application by the Microprocessor Unit. The MPEG Video Decoding Unit uses an MPEG Video Buffer to decode the MPEG Video stream. Output of the MPEG Video Decoding Unit is the RGB full motion video plane, which is provided to the External Video input of the Overlay and Mixer of the Video Decoder Model described in Chapter V.4.1.

Figure IX.4.9 MPEG Video Decoder Model



---

## IX.4 MPEG Video

---

### 4.6.1.2 Transfer Buffer

In the Transfer Buffer data are stored from the ISO 11172 stream in MPEG Video sectors from the selected CD-I channel. The data are stored in the Transfer Buffer by the CD-I Sector Processor or the Microprocessor Unit. The data are read by the MPEG Video Decoding Unit.

In case of a play from disc, the Transfer Buffer is a chain of at least 2 circularly linked PCL Buffers, where each buffer has room to store 1 MPEG Video sector (2324 bytes); see Chapter VII.2, Play Control List (PCL) Format.

The provision of the Transfer Buffer is the responsibility of the application. During the slow motion forward mode and the single picture forward mode the Transfer Buffer compensates for the response time of the CD-I player system. It is recommended to applications to define during these modes a Transfer Buffer with enough PCL buffers to store ISO 11172 data for a playback duration of at least one second. The application is responsible for controlling the data transfer from disc to the bitrate required for that specific slow motion/single picture forward mode.

### 4.6.1.3 MPEG Video Decoding Unit

The MPEG Video Decoding Unit reconstructs pictures from an MPEG Video stream. Input to the MPEG Video Decoding Unit is an ISO 11172 stream. Pictures are reconstructed from one MPEG Video stream in the multiplex of the ISO 11172 stream. From which MPEG video stream pictures are reconstructed is controlled from the application by the Microprocessor Unit. After reconstruction, the pictures are displayed in the full motion video plane at the output of the MPEG Video Decoding Unit. Within the MPEG parameter constraints described in Chapter IX.4.3, the MPEG Video Decoding Unit meets the specifications for an MPEG Video decoder from the ISO 11172 standard.

IX.4 MPEG Video

---

**4.6.1.4 MPEG Video Buffer**

The MPEG Video Buffer is used by the MPEG Video Decoding Unit for storage purposes; the MPEG Video Buffer may be required for decoding as well as for the conversion from the coded picture rate to the frame rate of the full motion video plane. The size of the MPEG Video Buffer depends on the implementation; only the minimum size is specified. The MPEG Video Buffer shall contain at least 4Mbit, i.e.  $4 \times 1024 \times 1024$  bits.

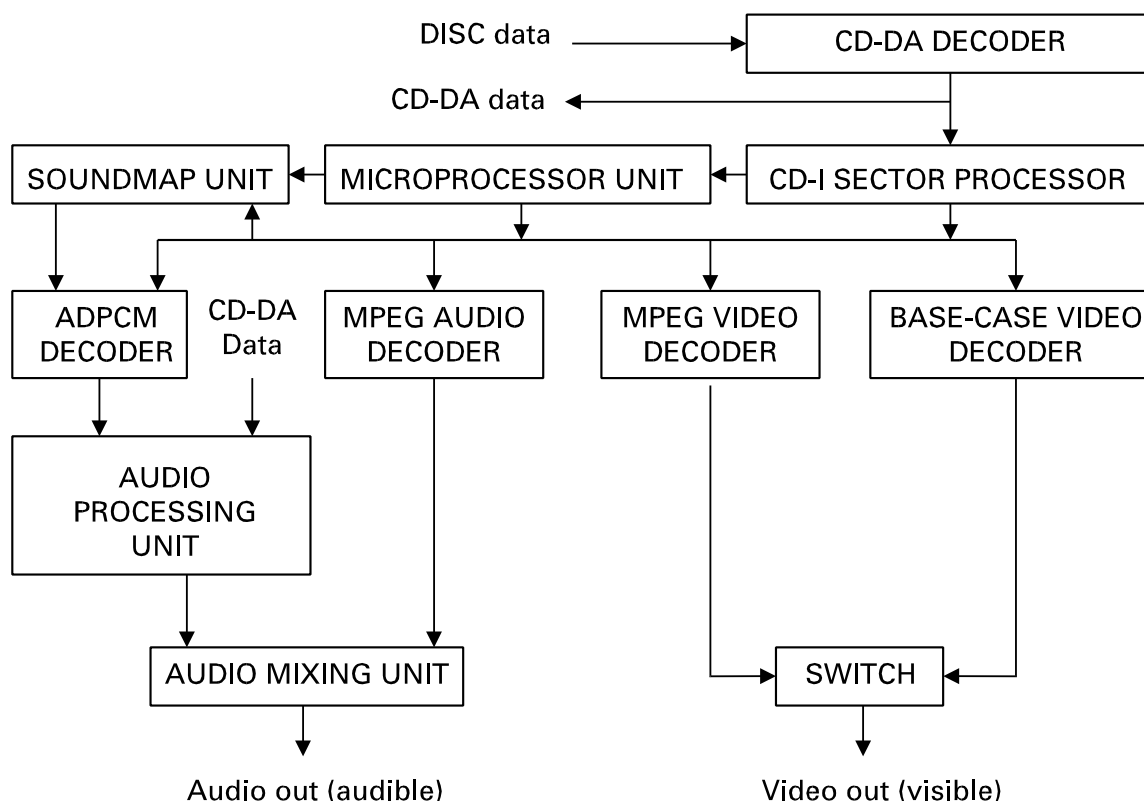
## IX.4 MPEG Video

## 4.6.2 Audio/Video Synchronization

## 4.6.2.1 Synchronization Model

A model to define synchronization between audio and video is depicted in figure IX.4.10. CD-I sectors from disc are provided by the CD-DA decoder to the CD-I Sector Processor. The CD-I Sector Processor provides the ADPCM Decoder, the MPEG Audio Decoder, the MPEG Video Decoder and the Base Case Video Decoder with the appropriate sectors. The CD-I Sector Processor may also transfer sectors to the Microprocessor Unit; the Microprocessor Unit may provide ADPCM soundmap data to the Soundmap Unit, MPEG Audio soundmap data to the MPEG Audio Decoder and MPEG Video videomap data to the MPEG Video Decoder. The output of the ADPCM decoder and CD-DA Data from CD-DA tracks are provided to the Audio Processing Unit.

Figure IX.4.10 Synchronization Model



IX.4 MPEG Video

---

The outputs of the Audio Processing Unit and the MPEG Audio decoder are mixed in the Audio Mixing Unit and presented as audible sound. The outputs of the MPEG Video Decoder and the Base Case Video Decoder are mixed into one video plane and presented as visible pictures.

To ensure synchronization between MPEG Video presentation and the other audio and video presentations, the delays are to be specified for each building block of the synchronization model. The delays for the building blocks from the Base Case System are specified in Chapter VIII. Some of these building blocks are more restrictively specified by the Full Motion System. In the Full Motion System the delays of the building blocks shall meet the following constraints:

## 1) CD-DA Decoder.

The delay caused by the CD-DA Decoder is to be constant. In a formula:

$$\tau(\text{cd-da}) = \text{constant}$$

## 2) CD-I Sector Processor.

For real-time sectors the delay  $\tau(\text{sp})$  caused by the CD-I Sector processor shall be less than or equal to 27 msec. This delay, specified for the Base Case System, has to meet an additional constraint. The delay  $\tau(\text{sp})$  can be split in a part  $\tau(\text{sp,constant})$  which is constant during a play and a part  $\tau(\text{sp,variable})$  which may vary in time during a play. In a Full Motion System the variable part  $\tau(\text{sp,variable})$  shall be less than or equal to one nominal sector period. In formula:

$$\begin{aligned} \tau(\text{sp}) &= \tau(\text{sp,constant}) + \tau(\text{sp,variable}) \\ \tau(\text{sp}) &\leq 27 \text{ msec.} \\ \tau(\text{sp,variable}) &\leq 13 \text{ msec.} \end{aligned}$$

## 3) Microprocessor Unit.

The delay due to the Microprocessor Unit is not specified explicitly, as such delay depends on processes activated by the application. Note however that the Microprocessor Unit is only involved in the transfer process of real-time sectors via an interrupt mechanism.

## 4) Soundmap Unit.

The Soundmap Unit holds ADPCM data, but does not contribute to the delay of ADPCM data.

## IX.4 MPEG Video

## 5) ADPCM Decoder.

The delay of ADPCM data caused by the ADPCM decoder and the Audio Processing Unit shall be less than or equal to 27 msec. This is the delay time between audio data input to the ADPCM Decoder and audible output. In a formula:

$$\tau(\text{adpcm}) \leq 27 \text{ msec.}$$

## 6) Audio Processing Unit.

For ADPCM data the delay of the Audio Processing Unit is specified implicitly already in  $\tau(\text{adpcm})$ . The delay of the Audio Processing Unit for CD-DA data shall be less than or equal to 27 msec. In a formula:

$$\tau(\text{apu,cd-da}) \leq 27 \text{ msec.}$$

## 7) MPEG Audio Decoder.

The delay  $\tau(\text{mpeg-audio})$  caused by the MPEG Audio Decoder can be split in a constant and a variable part  $\tau(\text{mpeg-audio,constant})$  and  $\tau(\text{mpeg-audio,variable})$ . The constant part shall be less than or equal to 40 msec. and the variable part shall be less than or equal to 27 msec. In formula:

$$\begin{aligned} \tau(\text{mpeg-audio}) &= \tau(\text{mpeg-audio,constant}) + \\ &\quad \tau(\text{mpeg-audio,variable}) \\ \tau(\text{mpeg-audio,constant}) &\leq 40 \text{ msec.} \\ \tau(\text{mpeg-audio,variable}) &\leq 27 \text{ msec.} \end{aligned}$$

## 8) MPEG Video Decoder.

The delay  $\tau(\text{mpeg-video})$  caused by the MPEG Video Decoder can be split in a constant and a variable part  $\tau(\text{mpeg-video,constant})$  and  $\tau(\text{mpeg-video,variable})$ . The constant part shall be less than or equal to 40 msec. and the variable part shall be less than or equal to 27 msec. Furthermore  $\tau(\text{mpeg-video,constant})$  shall be equal to  $\tau(\text{mpeg-audio,constant})$ . In formula:

$$\begin{aligned} \tau(\text{mpeg-video}) &= \tau(\text{mpeg-video,constant}) + \\ &\quad \tau(\text{mpeg-video,variable}) \\ \tau(\text{mpeg-video,constant}) &\leq 40 \text{ msec.} \\ \tau(\text{mpeg-video,constant}) &= \tau(\text{mpeg-audio,constant}) = \\ &\quad \tau(\text{mpeg,constant}) \\ \tau(\text{mpeg-video,variable}) &\leq 27 \text{ msec.} \end{aligned}$$

IX.4 MPEG Video

---

## 9) Base Case Video Decoder.

The delay caused by the Base Case Video Decoder between enabling an image for display and the image appearing on display shall be less than or equal to one display field period. In a formula:

$$\tau(\text{base-case-video}) \leq 20 \text{ msec.}$$

## 4.6.2.2 Synchronization with MPEG Audio

Play forward of MPEG Video streams synchronized with MPEG Audio is possible, both in case of play from disc and play from memory. A play from disc can be synchronized with a play from memory only, if the application issues the play from disc first.

A play forward of MPEG Video and MPEG Audio is fully synchronized, i.e. without any temporal presentation error, if the decoder system clock in a hypothetical System Target Decoder for MPEG Video and for MPEG Audio have a fixed offset  $\tau(\text{offset})$ . In a formula:

$$\text{dsc}(\text{std-mpeg-video}) - \text{dsc}(\text{std-mpeg-audio}) = \tau(\text{offset})$$

where

$\text{dsc}(\text{std-mpeg-video})$  is the value of the system clock in a System Target Decoder for MPEG Video at instant T.

$\text{dsc}(\text{std-mpeg-audio})$  is the value of the system clock in a System Target Decoder for MPEG Audio at instant T.

For each synchronized playback of MPEG Video and MPEG Audio, the value of  $\tau(\text{offset})$  shall be known by the application. The parameter  $\tau(\text{offset})$  has a resolution of 22.5 kHz (90,000 Hz / 4). In case of synchronized playback from disc of MPEG Video and MPEG Audio, the application shall apply the exact value of  $\tau(\text{offset})$  in encoding the SCR field in pack headers; see also Chapter IX.4.3.1.2. An MPEG Video stream can also be played back asynchronously, i.e. without a defined offset to MPEG Audio.

When both the MA\_Play and the MV\_Play function play from disc the application can send them in any order to the decoder. The play function that first receives data from the disc will supply synchronization information for the other play function.



IX.4 MPEG Video

---

When both the audio and the video map are of type host, it is the responsibility of the application that the play function of the map with the lowest start SCR value is sent first. The play function that must synchronize with the already started play should have a start SCR value that is greater than or equal to the current SCR value of the active play. When, due to the multi tasking behavior of the operating system, the current SCR of the running play is already greater than the start SCR at the time the second play function is started, this play function will stop. To prevent this, the application can set the syncpath parameter of the first play command that is issued to -2. This puts the play function in **wait mode** (see also IX-8, MV\_Play and MA\_Play). It will be activated as soon as the decoder receives a play command with the syncparameters set.

When both play functions play from disc and are grouped together by setting the syncpath parameter of the first play function to -2, the application should not start data retrieval from the CD until both playfunctions are send.

It is not possible to synchronize a play from disc to an active host play. To synchronize a hostplay to a play from disc, the play from disc should be started first.

After issuing a play command, decoding starts after a certain start-up delay. The length of this delay depends on the time required for the decoder system clock to reach the Time Stamp value of the picture or audio where the play is to start. In case of a synchronized play of MPEG Video and MPEG Audio, where either one or both are played from memory, the application shall issue each play command such, that the start-up delay has a positive value. Applications shall not issue play commands such that the decoding should have been started before the command is issued.

The total delay of MPEG Video data from disc has a constant component and a component that may vary in time; the same applies for the total delay of MPEG Audio data from disc. The constant components for MPEG Video data and MPEG Audio data have the same value:

$$\tau(\text{cd-da}) + \tau(\text{sp,constant}) + \tau(\text{mpeg,constant}).$$

The variable component of the MPEG Video delay from disc is equal to:

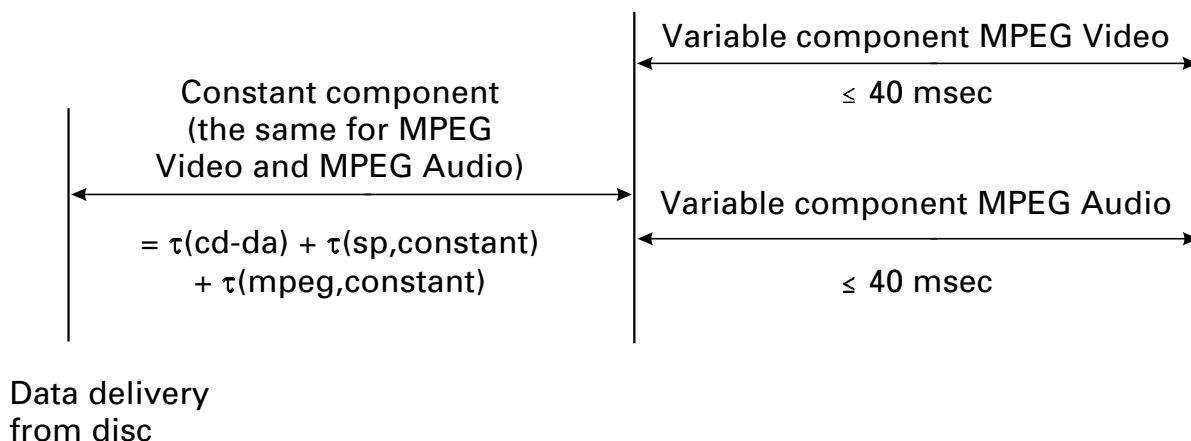
$$\tau(\text{sp,variable}) + \tau(\text{mpeg-video,variable}) \leq 13 + 27 = 40 \text{ msec.}$$

For MPEG Audio the variable component of the delay from disc is equal to:

$$\tau(\text{sp,variable}) + \tau(\text{mpeg-audio,variable}) \leq 13 + 27 = 40 \text{ msec.}$$

The maximum synchronization error in the presentation of MPEG Video and MPEG Audio is therefore 40 msec. See figure IX.4.11.

Figure IX.4.11 MPEG Video and MPEG Audio delays from disc



#### 4.6.2.3 Synchronization with ADPCM Audio

Play forward of MPEG Video can be combined with a play of ADPCM Audio from disc as well as from memory. The total delay of ADPCM data from disc within a Full Motion System has a constant and a variable component. The constant component is equal to:

$$\tau(\text{cd-da}) + \tau(\text{sp,constant}).$$

The variable component is equal to:

$$\tau(\text{sp,variable}) + \tau(\text{adpcm}) \leq 13 + 27 = 40 \text{ msec.}$$

For MPEG Video data from disc the variable component has the same range of 40 msec. For MPEG Video data the constant component is  $\tau(\text{mpeg,constant})$  larger. In case of a play forward of MPEG Video and ADPCM Audio, both from disc, the temporal presentation error may therefore be  $\tau(\text{mpeg,constant}) \pm 40$  msec. If the application knows the applied value of  $\tau(\text{mpeg,constant})$  in a CD-I System, the application can, in case of a ADPCM Audio play from disc, compensate for the presentation error due to  $\tau(\text{mpeg,constant})$  by using the Soundmap Unit as a compensating delay.

In case of an ADPCM play from the Soundmap Unit, it is the responsibility of the application not to activate processes that jeopardize the real-time tasks in the play process.

---

**IX.4 MPEG Video**

---

**4.6.2.4 Synchronization with CD-DA**

MPEG Video can be played from memory, while CD-DA is played from disc. CD-DA data is retrieved from disc with an inaccuracy in terms of playing time between +1 second and -1 second. It is therefore not possible to synchronize accurately a play of MPEG Video with a play of CD-DA .

**4.6.2.5 Synchronization with Base Case Video**

Synchronization of an MPEG Video play with Base Case Video is the responsibility of the application. To compensate for the delay of MPEG Video data, the application should delay the play of Base Case Video.

### 4.6.3 Synchronization between Events

To synchronize events in an application with events during an MPEG Video play, the application is, upon request, informed about the occurrence of MPEG Video events. Next to the two MPEG Video events described in Chapter IX.4.4.4.1, a number of MPEG Video events are defined to indicate:

- display of first intra-coded picture of a sequence starts;
- display of last picture of a sequence starts;
- display of first intra-coded picture of group of pictures starts;
- display of next coded picture starts;
- new sequence parameters are found at decoding;
- buffer underflow detected;
- error in input data detected;
- old PCL structure not in use any more.

An example is given in figure IX.4.12, where an MPEG Video stream starts with a Sequence Header (SEQ) and a Group of Pictures Header (GOP) and ends with a Sequence End Code (EOS). The display order of the pictures is indicated by means of an index  $i$ ; the picture type is indicated by means of a character I, B or P, where I, B and P mean I-type, P-type and B-type. See also Chapter IX.4.3.2.9. In this example upon request of the application the following events are generated:

- when picture 1 and picture 10 start being displayed, it is indicated that the first intra-coded picture from a sequence is displayed.
- when picture 12 starts being displayed, it is indicated that the last picture from a video sequence is displayed.
- when picture 1, picture 7 and picture 10 start being displayed, it is indicated that the first intra-coded picture from a group of pictures is displayed.
- when each next picture starts being displayed, it is indicated that a next picture is displayed.

A picture starts being displayed when at the output of the MPEG Video Decoder the first field is displayed which is dominantly derived from that picture. The event is generated before the active video starts at the output of the MPEG Video decoder of that first field, but after the vertical preceding active video.

IX.4 MPEG Video

---

Figure IX.4.12 **MPEG Video stream with decoded and displayed pictures**



Another option available to applications to control events is to read the value of the MPEG Video parameters "time code" and "temporal reference" at the MPEG Video Decoder. See ISO 11172 standard. The values read by the application refers to the picture which is displayed at the instant of reading. Furthermore the value of the MPEG Video decoder system clock can be read.

## IX.4 MPEG Video

## 4.6.4 Device Status Descriptor MPEG Video Decoder

The Device Status Descriptor (DSD, see Chapter VII.1.3) of the MPEG Video Decoder indicates the characteristics of an MPEG Video Decoder in a CD-I System. In the DSD the size of the MPEG Video Buffer (see Chapter IX.4.6.1.4) is indicated, as well as the value of  $\tau(\text{mpeg,constant})$ ; see Chapter IX.4.6.2.1.

The size of the MPEG Video Buffer in a CD-I System is specified as an integer number B, where B indicates an available buffer size of at least  $B * (1024 * 1024)$  bits. The minimum value of B is 4; the maximum value of B is not constrained.

In the DSD also the value of  $\tau(\text{mpeg,constant})$  is indicated; based on this value the application can introduce compensating delays for synchronization purposes. A three bit value can be coded in the DSD for  $\tau(\text{mpeg,constant})$ :  $k * 10$  msec., where  $k = 0 .. 7$ . The value of  $\tau(\text{mpeg,constant})$  in a CD-I system may be rounded for coding in the DSD; the maximum difference between the real value and the coded value is 5 msec.

For the MPEG Video Decoder the Device Status Descriptor is described in Figure IX.4.13.

Figure IX.4.13 MPEG Video Decoder DSD

**Device Type Code** 90

**Device Parameters**

<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
SZ#n	MPEG Video Buffer Size in Mbits	4
SD#n	$\tau(\text{mpeg,constant})$ in units of 10 msec. Range 0..7	0
DV#n	Device number (if multiple devices of that type)	0
SP#n	Number of Still Picture pages (see Chapter IX.6.6.4)	1
RV=string	FMV audio and video revision level. String="CTP1" : cartridge P1 String="CTP2" : cartridge P2	"CTP1"

## IX.4 MPEG Video

---

### 4.7 MPEG Video Data Re-arrangements

The application may re-arrange MPEG Video data e.g. to reduce the delay between successive MPEG Video loops from memory. See Chapter IX.4.4.5. Each MPEG Video stream in an ISO 11172 stream as well as each ISO 11172 stream resulting from re-arrangements by the application shall meet the specifications from the ISO 11172 standard. These streams shall further meet the constraints for MPEG parameters as specified in Chapter IX.4.3.

### 4.8 Extended MPEG Video Playing Time

In case of playing MPEG Video with lower bitrates, the play time can be extended in two ways. The first way is to play the CD-I track more than once with a different channel number or a different MPEG Video stream ID (see Chapter IX.4.3.1.5) selected for each scan. The second way is to apply large PCL Buffers as Transfer Buffer. See Chapter IX.4.6.1.2. One PCL Buffer is filled with MPEG Video data, while the play is from the other PCL Buffer. When all MPEG Video data from one PCL Buffer is read, that PCL Buffer is filled again with MPEG Video data while the play switches to the other. In such case the application is responsible to control the PCL Buffers.

### 4.9 Error Performance

Two error types in MPEG Video data are distinguished for error performance: flagged and un-flagged errors. The CD-I system may indicate flagged errors by means of an error flag. Un-flagged errors are errors which are not recognized; the decoder treats data with un-flagged errors as error-free data.

In case of a flagged error, the MPEG Video Decoder may apply the 'sequence error code' (see video part of ISO 11172) for error concealment. After encountering a flagged error, the MPEG Video Decoder may interrupt the decoding process until the next picture or the next group of pictures is encountered in the ISO 11172 stream. After an interrupt of the decoding process due an error, a previously correctly decoded picture is displayed till the decoding starts again. In case of a decoding interrupt, the application shall be informed about such event.

In case of un-flagged errors, the decoding proceeds with erroneous data; as a consequence, the decoding process may reach an undefined state and the video output of the decoder may be corrupted.

This page is intentionally left blank



---

**IX.5 MPEG Audio**

---

**IX.5 MPEG Audio****5.1 General MPEG Audio Encoding**

This section specifies encoding of audio by the Full Motion System into MPEG Audio data. Furthermore this section defines the MPEG Audio data structure, the decoding of MPEG Audio data, and the presentation of decoded MPEG Audio data as audible sound.

The Full Motion System encodes sound in mono or stereo into an MPEG Audio stream, applying the audio part of the MPEG standard ISO 11172 for coding audio-visual information. The Full Motion System applies the system part of ISO 11172 to multiplex one or more MPEG Audio streams into one ISO 11172 stream. In one ISO 11172 stream one to 32 MPEG Audio streams can be multiplexed. On a disc, one or more ISO 11172 streams with MPEG Audio data can be stored.

**5.2 MPEG Audio Data Structure****5.2.1 General**

An MPEG Audio stream is the coded representation of a mono, stereo or dual channel audio signal. Within CD-I one to 32 MPEG Audio streams can be multiplexed into one ISO 11172 stream.

**5.2.2 Files with MPEG Audio Data**

A CD-I file can contain one or more ISO 11172 streams with MPEG Audio data. Each ISO 11172 stream with MPEG Audio data is stored in an integer number of MPEG Audio sectors.

**5.2.3 Channels with MPEG Audio Data**

An ISO 11172 stream with MPEG Audio data can be part of only one CD-I channel. One CD-I channel may contain one or more ISO 11172 streams with MPEG Audio data. However, if to be played consecutively, the physical distance between MPEG Audio sectors from different ISO 11172 streams in the same CD-I channel in one file shall be at least 150 sectors, i.e. during a normal play the nominal time interval between reading MPEG Audio sectors from these different ISO 11172 streams will be at least two seconds.

---

**IX.5 MPEG Audio**

---

**5.2.4 MPEG Audio Sectors****5.2.4.1 General**

Each ISO 11172 stream with MPEG Audio data is stored in an integer number of MPEG Audio sectors. Each MPEG Audio sector cannot contain data from more than one ISO 11172 stream.

**5.2.4.2 Subheader Bytes****5.2.4.2.1 File Number**

The file number of MPEG Audio sectors conforms to the definition given in Chapter II.4.5.2, Chapter III.4.4 and in Appendix II.1.

**5.2.4.2.2 Channel Number**

The channel number of MPEG Audio sectors conforms to the definition from Chapter II.4.5.2, Chapter VII.2.2.3.2 (SS\_Play) and from Appendix II.2.

IX.5 MPEG Audio

---

## 5.2.4.2.3 Submode

In MPEG Audio sectors, the submode byte is encoded as shown in figure IX.5.1. Each MPEG Audio sector is a Form 2 sector of type Audio. The Form bit and the Audio bit are therefore set to "1"; both the Data bit and the Video bit are "0". The End of File bit, the Real-Time Sector bit, the Trigger bit and the End Of Record bit are coded conform Chapter II.4.5.3.

Figure IX.5.1 Encoding of the submode byte of an MPEG Audio sector

Bit Number	Bit Name	Bit Value
7	End Of File (EOF)	x
6	Real-Time sector (RT)	x
5	Form (F)	1
4	Trigger (T)	x
3	Data (D)	0
2	Audio (A)	1
1	Video (V)	0
0	End Of Record (EOR)	x

- Where:
- Bit Names are defined in Chapter II.4.5.3;
  - x indicates a don't care in the definition of the submode byte of an MPEG Audio sector;
  - Coding is in conformance with Chapter II.4.5.3

## IX.5 MPEG Audio

## 5.2.4.2.4 Coding Information

The coding information byte of an MPEG Audio sector is encoded as shown in figure IX.5.2. In each MPEG Audio stream parameters are indicating a.o. the coding mode (e.g. mono or stereo), the type of de-emphasis to be used, the sampling frequency and the bitrate. The bits in the coding information byte of an MPEG Audio sector are only used to indicate an MPEG Audio sector. There is no relation between the name of a bit, the coding of a bit and the parameters used for MPEG Audio coding.

Figure IX.5.2 Encoding of the coding information byte of an MPEG Audio sector

Bit Number	Bit Name	Bit Value
7	Reserved	0
6	Emphasis	1
5	Bits per Sample	1
4		1
3	Sample Frequency	1
2		1
1	Mono/Stereo	1
0		1

Where Bit Names are defined in Chapter IV.3.2.4.

Note: Emphasis, bits per sample, sample frequency and mono/stereo of MPEG Audio are indicated in the MPEG Audio stream. The bits in the coding information byte do not indicate any specific coding type.

## 5.2.4.3 MPEG Audio Sector Interleaving

The MPEG Audio sectors from one ISO 11172 stream are to be interleaved such, that in System Target Decoders, as defined in the MPEG standard, neither underflow nor overflow occurs.

---

**IX.5 MPEG Audio**

---

**5.3 MPEG Parameters****5.3.1 MPEG System Parameters****5.3.1.1 General**

The system part of the MPEG standard ISO 11172 is applied to store an ISO 11172 stream with MPEG Audio data in an integer number of MPEG Audio sectors. In each MPEG Audio sector one pack is stored; for the definition of a pack see the MPEG standard.

In an Audio sector 2304 bytes of audio data and a padded field containing 20 bytes are stored (see Chapter IV.3). A pack with MPEG Audio data consists of 2304 bytes; except for the sector containing the ISO 11172 End Code, in this sector the MPEG Audio pack consists of 2300 bytes. Within each pack an optional system header and one or more packets are stored; for the definition of a packet see the MPEG standard. A packet can be an audio packet, a padding packet, a reserved packet, a reserved data packet or a private packet. In case of an audio packet, it contains audio data from one MPEG audio stream. Each audio packet, including the packet header, consists of an even number of bytes. The example of an ISO 11172 stream with MPEG Video data given in figure IX.4.3, is also applicable for ISO 11172 streams with MPEG Audio data.

In ISO 11172 parameters are defined for the system part. The parameters are included in the pack header or in the packet header. Within the Full Motion Extension of CD-I some of these parameters are constrained in syntactical or semantical sense. In the following sections the constraints for MPEG Audio sectors to be applied on the parameters from the system part are defined.

**5.3.1.2 System Clock Reference**

In each MPEG Audio sector a pack is stored, consisting of a pack header, an optional system header and one or more packets with e.g. padding data or MPEG Audio data. The System Clock Reference (SCR) parameter in the pack header indicates the intended time of arrival of the last byte of the SCR field at the input of a hypothetical System Target Decoder (STD). See the MPEG standard. The SCR is measured in periods of 90 kHz.

Within the Full Motion system the frequency of the system clock in the STD has a fixed relation to the nominal rate at which sectors are delivered from CD-I disc:

$$f_{90000}(\text{std}) = 1200 * f_{75}(\text{sector})$$

where  $f_{90000}(\text{std})$  is the nominal frequency of the system clock in the STD and  $f_{75}(\text{sector})$  is the nominal rate at which sectors are delivered from disc.

As a consequence for real-time sectors from the same ISO 11172 stream, the difference between encoded SCR values in two sectors can be calculated from the distance between both sectors:

$$\text{SCR}(\text{sector } S_2) - \text{SCR}(\text{sector } S_1) = 1200 * (D + 1)$$

where D is the number of sectors on disc between sector  $S_1$  and sector  $S_2$ .

MPEG Audio streams from non-real-time sectors can be played from memory with only the nominal sector rate. In non-real-time MPEG Audio sectors from the same ISO 11172 stream the encoded SCR value and the distance on disc do not necessarily have a fixed relation. For non-real-time MPEG Audio sectors the above formula needs therefore modification to:

$$\text{SCR}(\text{sector } S_2) - \text{SCR}(\text{sector } S_1) = 1200 * N$$

where N is an integer number.

At the start of an ISO 11172 stream, the system clock may have any value allowed by the range of the SCR parameter. There is no fixed relation to the system clock in other ISO 11172 streams. System clocks in STD's for different ISO 11172 streams have a fixed but arbitrary offset due to the relation of the individual system clocks to the nominal sector delivery rate.

---

**IX.5 MPEG Audio**

---

**5.3.1.3 Mux Rate**

The Mux Rate parameter in the pack header indicates the rate at which the hypothetical System Target Decoder (STD) receives the bytes from the ISO 11172 stream in the MPEG Audio sector. Within the Full Motion system all ISO 11172 bytes from a sector arrive at a rate defined by the nominal period of that sector. The rate is found by multiplying the total number of bytes in the sector (2352) by the sector rate of 75 Hz. The Mux Rate field needs to be encoded with a value in units of 50 bytes. In the Mux Rate field of the pack header of an MPEG Audio sector therefore the value 3528 ( $= 75 * 2352 / 50$ ) is encoded.

Within the Full Motion System, the rate at which the bytes from an ISO 11172 stream are received by a STD, is constant for each byte in the ISO 11172 stream. Therefore in each MPEG Audio sector with data from one ISO 11172 stream the same value for the Mux Rate parameter is encoded.

**5.3.1.4 System Header Parameters**

All ISO 11172 streams within the Full Motion system are Constrained System Parameter Streams (CSPS). The CSPS Flag shall therefore be set to "1" in each System Header. In general MPEG Audio sectors are interleaved with other sectors, without meeting the requirements for fixed bitrate operation. Usually the Fixed Flag is therefore set to "0". In each MPEG Audio stream there is a constant relation between the audio sampling frequency and the system clock frequency in the System Target Decoder. In each System Header the System Audio Lock Flag is therefore set to "1".

Although no MPEG Video streams are included in MPEG Audio sectors, also the System Video Lock Flag is set to "1" in each System Header. The rate bound field shall not indicate a value of the mux rate parameter larger than allowed in the Full Motion system. The video bound field shall indicate that no MPEG Video streams are included in the multiplex. The value encoded in the STD buffer size bound field for MPEG Audio streams shall not indicate a value larger than 4096 Bytes.

---

**IX.5 MPEG Audio**

---

**5.3.1.5 Stream ID**

The Stream ID in the Packet Header specifies the type of stream stored in the packet. MPEG Audio sectors do not contain MPEG Video streams. Values of the Stream ID parameter indicating an MPEG video stream are therefore not allowed. All other values for the Stream ID parameter allowed by the MPEG standard can be used. However, MPEG Audio decoders within the Full Motion system will ignore packets with reserved streams as well as packets with private streams.

In case of a packet with an MPEG Audio stream, the Stream ID also indicates the number of the MPEG Audio stream. All Stream ID numbers allowed by the MPEG standard can be used.

**5.3.1.6 STD Buffer Size**

The STD Buffer Size is bounded by the constraints for CSPS streams in the MPEG standard. In the Packet Header of MPEG Audio streams therefore a STD Buffer Size shall be indicated which is smaller than or equal to  $4 * 1024$  Bytes.

**5.3.1.7 Time Stamps**

An MPEG Audio stream contains a series of one or more coded audio frames. Within the Full Motion system MPEG Audio streams are stored in packets. In each packet in which a coded audio frame commences, the Time Stamp is encoded in the packet header. A coded audio frame commences in a packet, if the first byte of the synchronization word of the audio frame is present in the packet data.

**5.3.1.8 Packet Rate**

The Rate at which packets shall arrive at the input of the hypothetical System Target Decoder (STD) is bounded to 300 Hz, as specified in the system part of the MPEG standard for CSPS streams. Next to the CSPS requirement for the packet rate, an additional requirement is to be met; applications shall not store more than four packets or system headers in one MPEG Audio sector.



---

**IX.5 MPEG Audio**

---

**5.3.2 MPEG Audio Parameters****5.3.2.1 General**

The Full Motion System codes audio corresponding to the audio part of the MPEG standard ISO 11172. After coding an MPEG Audio stream is available. An MPEG Audio stream is a series of one or more audio sequences. An audio sequence is a non-interrupted series of one or more audio frames. In an audio sequence the following MPEG Audio parameters shall not change:

- ID,
- Layer,
- Bit Rate Index and
- Sampling Frequency.

An audio sequence is interrupted if the presentation time of an audio frame is greater than the largest value permissible by the presentation time of the previous frame and the specified tolerance on the system clock frequency.

The coded representation of the next MPEG audio sequence in the MPEG Audio stream does not necessarily commence immediately after the coded representation of the previous audio sequence. The decoding and presentation of audio frames from the next MPEG audio sequence does not necessarily commence immediately after the decoding and presentation of audio frames from the previous MPEG audio sequence.

In the audio part of ISO 11172 the MPEG Audio Parameters are defined. The Full Motion System applies some constraints in syntactical or semantical sense to these parameters. In the following sections all constraints on parameters from the audio part of ISO 11172 are defined. On parameters not referred to, no specific constraints apply.

**5.3.2.2 ID**

The ID bit indicates the ID of the coding algorithm. In MPEG Audio streams within the Full Motion System the ID bit shall be set to "1", indicating that the MPEG Audio algorithm is applied.

IX.5 MPEG Audio

---

**5.3.2.3 Layer**

The two Layer bits indicate the applied MPEG Audio coding Layer. The Full Motion System supports Layer I and Layer II. Other Layers are not allowed.

**5.3.2.4 Protection Bit**

Within the Full Motion System the option can be used to add redundancy to the MPEG Audio stream by means of a CRC check.

## IX.5 MPEG Audio

## 5.3.2.5 Bit Rate Index

The Full Motion System supports all bitrate options provided by the audio part of the MPEG standard, with the exception of the 'free format', which is not supported. See also figure IX.5.3. In each audio sequence, the bitrate shall not change.

Figure IX.5.3 Available bitrate options for MPEG Audio streams

bitrate index	Layer I			Layer II		
	bitrate	mono	stereo	bitrate	mono	stereo
0000	not allowed			not allowed		
0001	32	yes	yes	32	yes	no
0010	64	yes	yes	48	yes	no
0011	96	yes	yes	56	yes	no
0100	128	yes	yes	64	yes	yes
0101	160	yes	yes	80	yes	no
0110	192	yes	yes	96	yes	yes
0111	224	yes	yes	112	yes	yes
1000	256	yes	yes	128	yes	yes
1001	288	yes	yes	160	yes	yes
1010	320	yes	yes	192	yes	yes
1011	352	yes	yes	224	no	yes
1100	384	yes	yes	256	no	yes
1101	416	yes	yes	320	no	yes
1110	448	yes	yes	384	no	yes
1111	not allowed			not allowed		
Notes:	bitrate in kbit per second; stereo refers to dual channel mode, independent stereo and intensity stereo mode; mono refers to single channel mode.					

---

**IX.5 MPEG Audio**

---

**5.3.2.6 Sampling Frequency**

The Full Motion System supports a Sampling Frequency of MPEG Audio streams of 44.1 kHz. Other Sampling Frequencies are not allowed.

**5.3.2.7 Private Bit**

Within the Full Motion System the Private Bit is reserved for future use. Until further notice, applications shall set the Private Bit to "0".

**5.3.2.8 Mode and Mode Extension**

The Mode and Mode Extension Bits are used to indicate the mode (stereo, joint stereo, dual channel or single channel) and which sub-bands are coded in intensity stereo. All options for Layer I and Layer II provided by the audio part of ISO 11172 are supported by the Full Motion System.

**5.3.2.9 Copyright**

Applications may indicate by means of the Copyright Bit whether there is copyright on the MPEG Audio stream.

**5.3.2.10 Original/home**

Applications may indicate by means of the Original/home Bit whether an MPEG Audio stream is an original or a copy.

**5.3.2.11 Emphasis**

Within the Full Motion System two Emphasis options may be used: no emphasis and 50/15  $\mu$ sec. emphasis. Other options, including CCITT J.17 emphasis, are not allowed.

IX.5 MPEG Audio

---

**5.3.2.12 Ancillary Data**

Ancillary Data is application specific data. MPEG Audio streams may contain Ancillary Data, but Full Motion Audio Decoders will not interpret such data.

**5.3.3 Access to MPEG Audio streams**

Applications can start decoding of an MPEG Audio stream at each audio frame.

**5.3.4 Length of an MPEG Audio stream**

The minimum length of an MPEG Audio stream is one audio frame. The maximum length is only limited by the available disc capacity.

---

**IX.5 MPEG Audio**

---

**5.4 Presentation of MPEG Audio****5.4.1 Mixing with CD-I Base Case Audio**

The MPEG Audio Decoder reconstructs audio from the MPEG Audio stream which is input to the decoder. MPEG Audio-channel zero (ch=0) is the Left output signal and MPEG Audio-channel one (ch=1) is the Right output signal from the MPEG Audio decoder. These output signals can be mixed with the Left and Right output signals from the Base Case Audio Processing Unit. The output signals from the MPEG Audio Decoder are made available as External Audio to the Audio Decoder Model. See Chapter IV.5.1. The Left and Right outputs are connected to the corresponding Left and Right inputs of the Audio Mixing Unit, described in Chapter IV.6.3.

**5.4.2 Mixing Control Functions**

The MPEG Audio Decoder contains an Audio Mixing Control Unit conform to the description in Chapter IV.6.3, the Audio Mixing Unit processes the Left and Right outputs of the MPEG Audio Decoder by means of four attenuators between left-in and left-out, left-in and right-out, right-in and right-out and right-in and left-out. The attenuators are controlled in steps of 1 dB. See Chapter VIII.4.2.1. The resulting left-out and right-out signals are added to the Audio Processing Unit output signals (left to left and right to right) to obtain the final left and right output signals. It is the responsibility of the application to define attenuators control values such that an output level does not exceed any of the input levels.

---

**IX.5 MPEG Audio**

---

**5.4.3 Playback Control Functions****5.4.3.1 General**

An MPEG Audio Decoder can playback an MPEG Audio stream multiplexed into an ISO 11172 stream. Simultaneously an MPEG Audio Decoder can play only one MPEG Audio stream from one ISO 11172 stream. During a playback mode an application shall not switch to another ISO 11172 stream. Switching to another stream is only possible by first aborting the playback mode of the current stream.

The application can switch between different MPEG Audio streams within an ISO 11172 stream. After switching to a new stream, the MPEG Audio decoder will not accept data from the previous stream anymore. The decoding of the new MPEG Audio stream will commence at the audio frame immediately after the first byte of the synchronization word encountered in that stream. The decoding of the new stream has the highest priority; the MPEG Audio decoder may decide not to finalize completely the decoding of the previous stream.

An MPEG Audio stream is a series of one or more audio sequences; see also Chapter IX.5.3.2.1. The MPEG Audio decoder may mute the first audio frame of a new audio sequence.

If the Protection Bit (see Chapter IX.5.3.2.4) is set to "1", i.e. no redundancy has been added to the audio sequence, the MPEG Audio decoder may mute the first decoded audio frame of a play back mode.

When at the MPEG Audio Decoder no further decoded audio frames are available for presentation, the MPEG Audio Decoder shall mute its output, until decoded audio frames from further MPEG Audio data are to be presented again.

When during a playback mode the ISO 11172 stream ends, i.e. the ISO 11172 End Code is encountered in the stream, the MPEG Audio playback mode remains active. Upon request of the application, the MPEG Audio Decoder informs the application during each playback mode on the occurrence of this MPEG Audio event:

- an ISO 11172 End Code (see system part of MPEG standard) is encountered in the ISO 11172 stream at the input of the MPEG Audio Decoder.

Based on above event, the application can control the MPEG Audio Decoder.

## IX.5 MPEG Audio

---

For MPEG Audio two playback modes are distinguished: play forward and mute. For both modes the functionality is described in the following sections. The described functionality is provided for playback from disc as well as from system memory, unless stated otherwise.

### 5.4.3.2 Access Address in MPEG Audio Stream

A playback of an MPEG Audio stream can start at each audio frame in the stream. In case of a play from disc, the application can indicate at which audio frame the playback is to start by addressing an access position by means of an MPEG File Pointer and an MPEG Audio Pointer. For byte indices in MPEG Audio sectors see e.g. Chapter II.4.8.1. and Chapter IV.3.3.

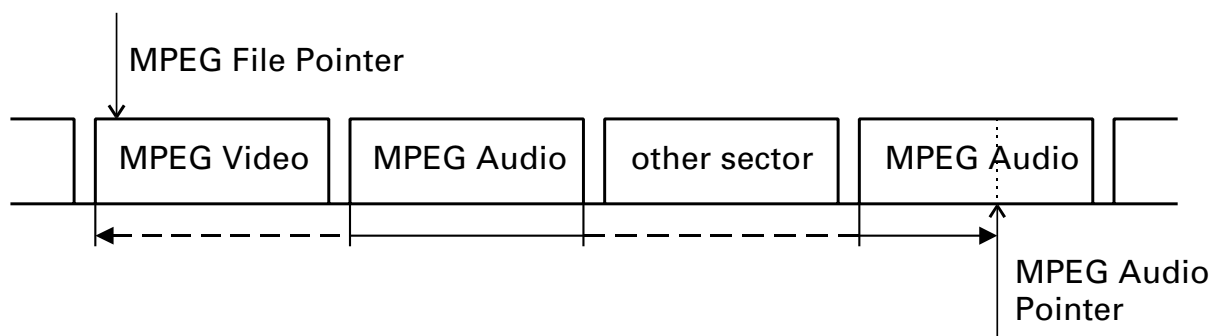
The MPEG File Pointer addresses the first sector to be retrieved from disc. This sector may be related to playback requirements of e.g. MPEG Video, Base Case Video or ADPCM Audio. The sector addressed by the MPEG File Pointer may therefore be any sector from the file with the MPEG Audio stream to be played. Applications should take into account that file pointers use a sector size of 2048 bytes to calculate a position within a file; see Chapter VII.2.2.3.2, the description of a function to issue a seek command to the disc drive (SS\_Seek).

The MPEG Audio Pointer addresses the first byte to be accessed in the ISO 11172 stream. The MPEG Audio Pointer shall address the first byte of a pack start code or the first byte of the synchronization word of an audio frame. See system part and audio part of ISO 11172.

In case of an access from disc, the MPEG Audio Pointer addresses the first byte of the access by indicating the index of that byte within the ISO 11172 stream from the selected CD-I channel. Note that one CD-I channel may contain sub-sequent ISO 11172 streams with MPEG Audio data. See Chapter IX.5.2.3. The first byte of the ISO 11172 stream retrieved from disc in the selected CD-I channel has index zero. See also the example in figure IX.5.4. Applications should take into account that MPEG Audio sectors contain 2304 ISO 11172 bytes.



Figure IX.5.4 Example MPEG File Pointer and MPEG Audio Pointer



MPEG File Pointer points to MPEG Video sector. Data retrieval from disc will therefore start with this sector.

MPEG Audio Pointer indicates the index of a byte in the ISO 11172 stream from the selected CD-I channel. If this Pointer has the value zero, the access will start at the pack header from the first MPEG Audio sector retrieved from disc in the selected CD-I channel.

In this example the MPEG Audio Pointer indicates the first byte of the synchronization word of an audio frame. The MPEG Audio Pointer equals 3548; therefore the first accessed byte of the MPEG Audio stream is the 1245th byte ( $3548 - 2304 + 1 = 1245$ ) in the second MPEG Audio sector from the selected CD-I channel retrieved from disc.

In case of an access from memory, only the MPEG Audio Pointer is used. In such case, the MPEG Audio Pointer indicates the index of a byte from the ISO 11172 stream with MPEG Audio data stored in memory. The first byte of such ISO 11172 stream has index zero.

Decoding will start at the first audio frame encountered in the MPEG Audio stream from the byte referred to by the MPEG Audio Pointer. An audio frame is encountered in the stream if the first byte of its synchronization word is encountered.

### 5.4.3.3 Play Forward

In the play forward mode an MPEG Audio stream is decoded and played back in normal speed forward. A play forward can be paused and continued. A play forward can start at each audio frame; see Chapter IX.5.3.3. and IX.5.4.3.2. From the mute mode a play forward will start at the first audio frame encountered in the MPEG Audio stream.

### 5.4.3.4 Mute

In the mute mode the output from the MPEG Audio Decoder is muted. Entering and exiting the mute mode is asynchronous with the decoding process of audio frames.

In case of a synchronous play back of MPEG Video and MPEG Audio, the output from the MPEG Audio decoder is muted when the MPEG Video decoder is in a playback mode different from Normal Speed. After changing the MPEG Video decoder to normal speed, audio will be de-muted again and synchronization will be restored.

---

**IX.5 MPEG Audio**

---

**5.4.4 MPEG Audio Memory Maps**

The Full Motion System supports playback of an ISO 11172 stream, or a continuous part thereof, from memory. An MPEG Audio stream multiplexed into an ISO 11172 stream, or a part thereof, shall be played from memory only, if the ISO 11172 stream meets the specifications from the ISO 11172 standard, as well as the constraints for MPEG parameters as specified in Chapter IX.5.3. The map in memory of an ISO 11172 stream played from memory, shall commence with the first byte of a pack start code.

Play back of MPEG Audio is possible from disc or from memory with the same playback functionality. Play from disc and play from memory are mutually exclusive functions. During an MPEG Audio play from memory, the bitrate for the ISO 11172 stream is bounded by the constraints specified in Chapter IX.5.3.1.3.

A play of an MPEG Audio map starts with the byte in the ISO 11172 stream indicated by the MPEG Audio Pointer. The MPEG Audio Pointer indicates the first byte of the synchronization word of an audio frame or the first byte of a pack\_start\_code. See Chapter IX.5.4.3.2. The MPEG Audio stream can be played from the first audio frame in the stream until the last audio frame included in the stream.

With memory maps, MPEG Audio loops can be created, by means of which MPEG Audio sequences are repeated. The number of loops to be played is specified by the application. The first and last audio frame in the loop are defined by the application by means of two MPEG Audio Pointers. The MPEG Audio Pointer defining the first audio frame in the loop indicates the first byte of a pack start code or the first byte of the synchronization word of the first audio frame in the loop. See Chapter IX.5.4.3.2. The MPEG Audio Pointer defining the last audio frame indicates the byte following the last byte of the coded representation of the last audio frame in the loop.

An MPEG Audio loop ends when all audio frames are decoded from the selected MPEG Audio stream between both MPEG Audio Pointers to the ISO 11172 stream. At the start of the next loop, there will be some delay before the first audio frame from the next loop is decoded. The length of this delay depends on the timing characteristics of the stream, especially on the encoded values in the first SCR field and the first Time Stamp field in the loop. See MPEG standard; see also Chapter IX.5.3.1.2 and Chapter IX.5.3.1.7. During this delay, the MPEG Audio Decoder will mute its output.

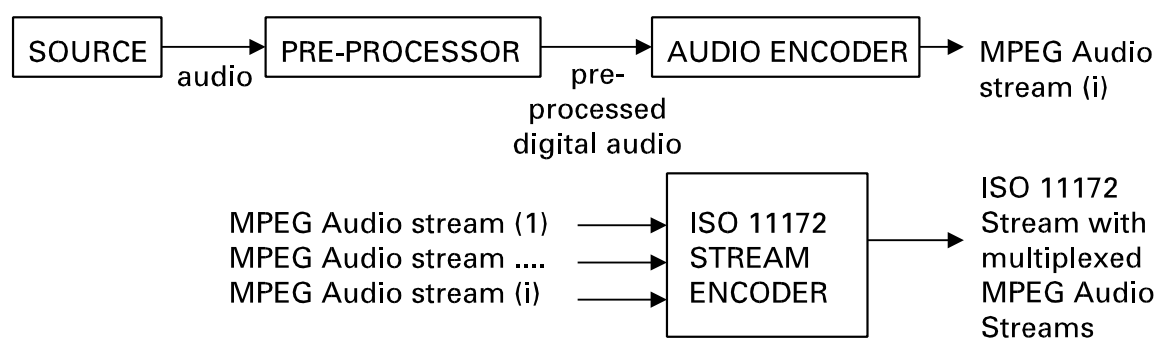
## IX.5 MPEG Audio

## 5.5 MPEG Audio Data Encoding

## 5.5.1 MPEG Audio Encoding Model

The MPEG Audio encoding model is depicted in figure IX.5.5. The source, e.g. a tape recorder, is delivering the audio signal to the pre-processing; output of the pre-processor are the audio data which are to be coded. At the output of the audio encoder, a coded MPEG Audio stream is available at the required bitrate. Finally one or more MPEG Audio streams are input to the ISO 11172 stream encoder, which produces an ISO 11172 stream at the format required for storage in an MPEG Audio sector.

Figure IX.5.5 MPEG Audio Encoding Model



## 5.5.2 Pre-processor

Especially in case of low bitrates, the MPEG Audio encoding may require a pre-processed audio input to optimize the audio quality of the result. One simple option of the pre-processor may be to reduce the bandwidth of the audio data. In case of an analog input signal, A to D conversion will be included in the pre-processor, as the Audio Encoder requires a digital input format.

## 5.5.3 Audio Encoder

The Audio Encoder produces an MPEG Audio stream from the digital audio data at the input. Within the flexibility provided by ISO 11172 and within the constraints described in Chapter IX.5.3.2, it is at the discretion of the implementation which encoding techniques are applied. The application defines which values are applied to MPEG Audio parameters during encoding.

IX.5 MPEG Audio

---

**5.5.4 ISO 11172 Stream Encoder**

The ISO 11172 stream encoder multiplexes one or more MPEG Audio streams into one ISO 11172 stream. The format of the produced ISO 11172 stream meets the constraints described in Chapter IX.5.3.1. The multiplex depends on which sectors are allocated as MPEG Audio sectors. The MPEG Audio sector allocation is therefore provided before multiplexing or is specified by the ISO 11172 stream encoder. The applied multiplex technique is at the discretion of the implementation.

IX.5 MPEG Audio

---

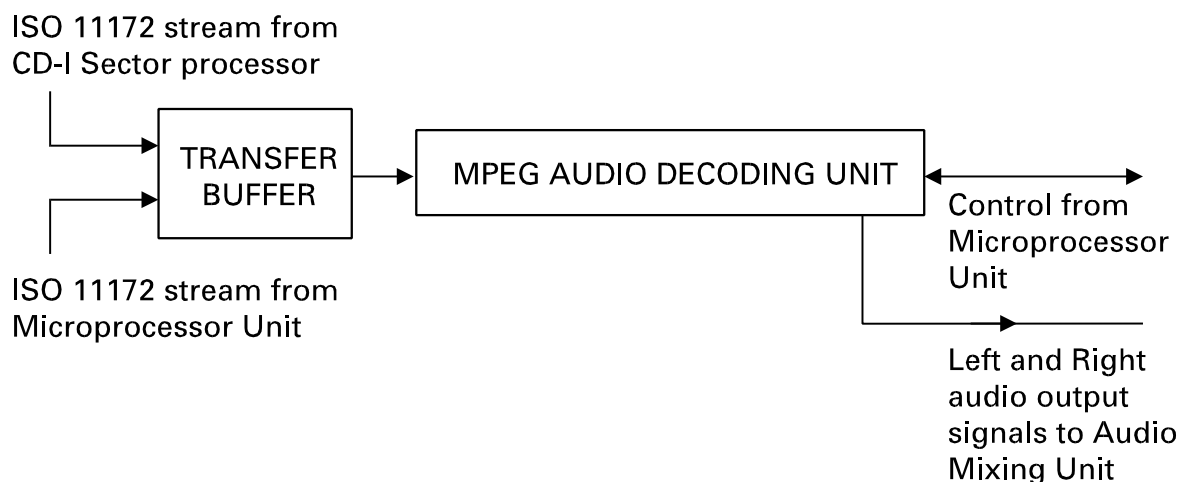
## 5.6 MPEG Audio Decoding

## 5.6.1 MPEG Audio Decoder

## 5.6.1.1 MPEG Audio Decoding Model

The MPEG Audio decoding model is given in figure IX.5.6. The CD-I Sector Processor (see Chapter II.6.1) or the Microprocessor Unit is delivering an ISO 11172 stream from MPEG Audio sectors in the selected CD-I channel. The ISO 11172 stream is stored in a Transfer Buffer. Subsequently, the MPEG Audio Decoding Unit reads the ISO 11172 stream from the Transfer Buffer to decode one of the MPEG Audio streams in the ISO 11172 stream. Which MPEG Audio stream is decoded is controlled by the Microprocessor Unit. Output of the MPEG Audio Decoding Unit are Left and Right audio signals, which are provided to the External Audio inputs of the Audio Mixing Unit, described in Chapter IV.6.3. See also Chapter IX.5.4.1.

Figure IX.5.6 MPEG Audio Decoding Model



---

**IX.5 MPEG Audio**

---

**5.6.1.2 Transfer Buffer**

In the Transfer Buffer data are stored from the ISO 11172 stream in MPEG Audio sectors from the selected CD-I channel. The data are stored in the Transfer Buffer by the CD-I Sector Processor or the Microprocessor Unit. The data are read by the MPEG Audio Decoding Unit.

In case of a play from disc, the Transfer Buffer is a chain of at least 2 circularly linked PCL Buffers, where each buffer has room to store one sector (2304 bytes); see Chapter VII.2, Play Control List (PCL) Format.

The provision of the Transfer Buffer is the responsibility of the application.

**5.6.1.3 MPEG Audio Decoding Unit**

The MPEG Audio Decoding Unit reconstructs audio signals from an MPEG Audio stream. Input to the MPEG Audio Decoding Unit is an ISO 11172 stream. The MPEG Audio Decoding Unit reconstructs simultaneously audio from only one MPEG Audio stream in the multiplexed ISO 11172 stream. From which MPEG audio stream audio is reconstructed is controlled from the application by the Microprocessor Unit. After reconstruction, the Left and Right audio output signals are available. Within the MPEG parameter constraints described in Chapter IX.5.3, the MPEG Audio Decoding Unit meets the specifications for an MPEG Audio decoder from the ISO 11172 standard.

## 5.6.2 Audio/Video Synchronization

### 5.6.2.1 Synchronization Model

To define synchronization between MPEG Audio and other audio and video presentations the same Synchronization Model is applied as for MPEG Video. This model is described in Chapter IX.4.6.2.1 and depicted in figure IX.4.10.

### 5.6.2.2 Synchronization with MPEG Video

Play forward of MPEG audio streams synchronized with MPEG video is possible, both in case of play from disc and from system memory. In case of play from system memory, the application is responsible not to activate processes that may jeopardize the real-time tasks of the play.

For the description and the specification of synchronizing playback of MPEG Audio and MPEG Video streams see Chapter IX.4.6.2.2.

### 5.6.2.3 Synchronization with ADPCM Audio

Play forward of MPEG Audio can be combined with a play of ADPCM Audio from disc as well as from memory. The requirements for a synchronized playback between MPEG Audio and ADPCM Audio are the same as for a synchronized playback between MPEG Video and ADPCM Audio. See Chapter IX.4.6.2.3.

### 5.6.2.4 Synchronization with CD-DA

MPEG Audio can be played from memory, while CD-DA is played from disc. CD-DA data is retrieved from disc with an inaccuracy in terms of playing time between +1 second and -1 second. It is therefore not possible to synchronize accurately a play of MPEG Audio with a play of CD-DA .

### 5.6.2.5 Synchronization with Base Case Video

Synchronization of an MPEG Audio play with Base Case Video is the responsibility of the application. To compensate for the delay of MPEG Audio data, the application should delay the play of Base Case Video.



IX.5 MPEG Audio

---

**5.6.3 Synchronization between Events**

For synchronization between events in an application and events during an MPEG Audio play, the application is, upon request, informed about the occurrence of MPEG Audio events. Next to the MPEG Audio event described in Chapter IX.5.4.3.1, a number of MPEG Audio event are defined:

- presentation of a (next) audio frame starts;
- decoder started decoding;
- buffer underflow occurred;
- decoding of other MPEG Audio stream started.

An audio frame starts being presented when the output signals of the MPEG Audio decoder start carrying information which is dominantly derived from that audio frame. The event is generated within a period starting 10 msec. before the audio frame starts being presented and lasting until 10 msec. after the start of presentation of the audio frame.

Another option available to applications to control events is to read the value of the MPEG Audio decoder system clock. To achieve synchronization between MPEG Audio events and other events is the responsibility of the application.

**5.6.4 Device Status Descriptor MPEG Audio Decoder**

A Device Status Descriptor (DSD, see Chapter VII.1.3) of a device indicates the characteristics of the device in a CD-I System. For the MPEG Audio Decoder the DSD is described in Figure IX.5.7.

Figure IX.5.7 **MPEG Audio Decoder DSD****Device Type Code** 91**Device Parameters**

<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
SD#n	$\tau(\text{mpeg,constant})$ in units of 10 msec. Range 0..7	0
DV#n	Device number (if multiple devices of that type)	0

---

**IX.5 MPEG Audio**

---

**5.7 MPEG Audio Data Re-arrangements**

The application may re-arrange MPEG Audio data e.g. to reduce the delay between successive MPEG Audio loops from memory. See Chapter IX.5.4.4. Each MPEG Audio stream in an ISO 11172 stream as well as each ISO 11172 stream resulting from re-arrangements by the application shall meet the specifications from the ISO 11172 standard. These streams shall further meet the constraints for MPEG parameters as specified in Chapter IX.5.3.

**5.8 Extended MPEG Audio Playing Time**

The play time of MPEG Audio can be extended in two ways, similar as for ADPCM Audio. See Chapter IV.7. The first way is to play the CD-I track more than once with a different channel number or with another MPEG Audio stream ID (see Chapter IX.5.3.1.5) selected for each scan; both cases are multiple plays. The second way is to apply large PCL Buffers as Transfer Buffer. See Chapter IX.5.6.1.2. One PCL Buffer is filled with MPEG Audio data, while the play is from the other PCL Buffer. When all MPEG Audio data from one PCL Buffer is read, that PCL Buffer is filled again with MPEG Audio data while the play switches to the other. In such case the application is responsible to control the PCL Buffers.

**5.9 Error Performance**

Two error types in MPEG Audio data are distinguished for error performance: flagged and un-flagged errors. A CD-I system may indicate flagged errors by means of an error flag. Un-flagged errors are errors which are not recognized; the decoder treats data with un-flagged errors as error-free data.

In case of a flagged error, the MPEG Audio Decoder may apply error concealment, may mute its output signals and may interrupt the decoding process until an audio frame without flagged errors is encountered.

In case of un-flagged errors, the decoding proceeds with erroneous data; as a consequence, the decoding process may reach an undefined state and the audio output signals of the decoder may be corrupted.

## IX.6 MPEG Still Picture

---

### IX.6 MPEG Still Picture

#### 6.1 General MPEG SP Encoding

In this section the encoding of still pictures by the Full Motion System is specified. Furthermore this section defines the MPEG SP data structure, the decoding of coded MPEG SP data back to a still picture, and the display of reconstructed pictures.

The Full Motion System applies the video part of the MPEG standard ISO 11172 to code still pictures. One or more still pictures can be coded into one MPEG SP stream. The Full Motion System applies the system part of ISO 11172 to multiplex one or more coded MPEG SP streams into one multiplexed ISO 11172 stream. In one ISO 11172 stream with MPEG SP data, one to sixteen MPEG SP streams can be multiplexed. It is not allowed to multiplex MPEG SP streams and MPEG Video streams, as defined in Chapter IX.4, into one ISO 11172 stream. On a disc, one or more ISO 11172 streams with MPEG SP data can be stored.

#### 6.2 MPEG SP Data Structure

##### 6.2.1 General

An MPEG SP stream is the coded representation of a sequence of one or more still pictures. One to sixteen MPEG SP streams can be multiplexed into one ISO 11172 stream.

##### 6.2.2 Files with MPEG SP Data

A CD-I file can contain one or more ISO 11172 streams with MPEG SP data. Each ISO 11172 stream with MPEG SP data is stored in an integer numbers of MPEG SP sectors.

### 6.2.3 Channels with MPEG SP Data

An ISO 11172 stream with MPEG SP data can be part of only one CD-I channel. One CD-I channel may contain one or more ISO 11172 streams with MPEG SP data. If to be played consecutively, the physical distance between MPEG SP sectors from different ISO 11172 streams in the same CD-I channel in one file shall be at least 150 sectors, i.e. at normal play speed, the nominal time interval between reading MPEG SP sectors from these different ISO 11172 streams will be at least two seconds.

### 6.2.4 MPEG SP Sectors

#### 6.2.4.1 General

Each ISO 11172 stream with MPEG SP data is stored in an integer numbers of MPEG SP sectors. Each MPEG SP sector cannot contain data from more than one ISO 11172 stream.

#### 6.2.4.2 Subheader Bytes

##### 6.2.4.2.1 File Number

The file number of MPEG SP sectors conforms to the definition given in Chapter II.4.5.2, Chapter III.4.4 and in Appendix II.1.

##### 6.2.4.2.2 Channel Number

The channel number of MPEG SP sectors conforms to the definition from Chapter II.4.5.2, Chapter VII.2.2.3.2 (SS\_Play) and from Appendix II.2.

IX.6 MPEG Still Picture

---

## 6.2.4.2.3 Submode

In MPEG SP sectors, the submode byte is encoded as shown in figure IX.6.1. Each MPEG SP sector is a Form 2 sector of type Video. The Form bit and the Video bit are therefore set to "1"; both the Data bit and the Audio bit are "0". The End of File bit, the Real-Time Sector bit, the Trigger bit and the End Of Record bit are coded conform Chapter II.4.5.3.

Figure IX.6.1 Encoding of the submode byte of an MPEG SP sector

Bit Number	Bit Name	Bit Value
7	End Of File (EOF)	x
6	Real-Time sector (RT)	x
5	Form (F)	1
4	Trigger (T)	x
3	Data (D)	0
2	Audio (A)	0
1	Video (V)	1
0	End Of Record (EOR)	x

- Where:
- Bit Names are defined in Chapter II.4.5.3;
  - x indicates a don't care in the definition of the submode byte of an MPEG SP sector;
  - Coding is in conformance with Chapter II.4.5.3

## IX.6 MPEG Still Picture

## 6.2.4.2.4 Coding Information

The coding information byte of an MPEG SP sector is encoded as shown in figure IX.6.2. Each MPEG SP sector is not application specific. The ASCF bit is therefore "0". The Even/Odd Lines Flag (EOLF) bit is not needed for error concealment and is therefore set to "0". The Resolution bits indicate a SP picture size. The Coding bits indicate MPEG coding.

Figure IX.6.2                    **Encoding of the coding information byte of an MPEG SP sector**

Bit Number	Bit Name	Bit Value
7	ASCF	0
6	EOLF	0
5	Resolution	0
4		1
3	Coding	1
2		1
1		1
0		1

- Where:
- Bit Names are defined in Chapter V.6.3.1;
  - Resolution bits indicate SP resolution;
  - Coding bits indicate MPEG coding.

## 6.2.4.3 MPEG SP Sector Interleaving

The MPEG SP sectors from one ISO 11172 stream are to be interleaved such, that in System Target Decoders, as defined in the MPEG standard, neither underflow nor overflow occurs.

---

**IX.6 MPEG Still Picture**

---

**6.3 MPEG Parameters****6.3.1 MPEG System Parameters**

To store an ISO 11172 stream with MPEG SP data in an integer number of sectors the system part of the MPEG standard ISO 11172 is applied. In each MPEG SP sector one pack is stored; for the definition of a pack see the MPEG standard. A pack consists normally of 2324 bytes, corresponding to the number of data bytes in a Video sector. The final pack of an ISO 11172 stream contains 2320 bytes, to allow for the ISO 11172 End Code of 4 bytes. Within each pack an optional system header and one or more packets are stored; for the definition of a packet see the MPEG standard. A packet can be a video packet, a padding packet, a reserved packet, a reserved data packet or a private packet. If the packet is a video packet, it contains data from one MPEG SP stream. Each video packet, including the packet header, consists of an even number of bytes. The first byte of a video packet must be located on a byte with an even index number within a sector (see Chapter II.4.1.2). The number of packet data bytes in each packet contained by an MPEG SP sector is even. The example of an ISO 11172 stream with MPEG Video data given in figure IX.4.3, is also applicable for ISO 11172 streams with MPEG SP data.

In ISO 11172 parameters are defined for the system part. The parameters are included in the pack header or in the packet header. Within the Full Motion System some of these parameters are constrained in syntactical or semantical sense. The constraints on the parameters from the system part to be applied for MPEG SP sectors are the same constraints which are to be applied for MPEG Video sectors (see Chapter IX.4.3.1) with the following exception. The Decoding Time Stamp field shall be encoded with the value equal to the arrival time in the System Target Decoder of the last byte of the Sequence End Code, which concludes the coded representation of the still picture. The Presentation Time Stamp is encoded consistently with the encoding of the Decoding Time Stamp, taking into account the encoded value of the picture rate.

---

**IX.6 MPEG Still Picture**

---

**6.3.2 MPEG Video Parameters****6.3.2.1 General**

Still pictures are coded corresponding to the video part of the MPEG standard ISO 11172. After coding an MPEG SP stream is available. An MPEG SP stream is a series of one or more MPEG video sequences. Each such MPEG video sequence consists of one still picture. The coded representation of the next MPEG video sequence in the MPEG SP stream does not necessarily commence immediately after the coded representation of the previous video sequence.

In the video part of ISO 11172 the MPEG Video parameters are defined. Within the Full Motion System some of these parameters are constrained in syntactical or semantical sense. In the following sections for MPEG SP streams all constraints on parameters from the video part are defined. On parameters not referred to, no specific constraints apply.

**6.3.2.2 Picture Size**

By means of the MPEG video standard rectangular pictures can be coded. The picture size is defined at encoding. The total picture area which can be coded is limited, but can be used in a flexible way. For example wide and low pictures can be coded as well as high and narrow pictures. The minimum size of a coded picture is one pixel by one line. The maximum area is 384 pixels by 576 lines.

The horizontal size shall be less than or equal to 768 pixels. The vertical size shall be less than or equal to 576 lines. The total picture area shall be less than or equal to 864 macroblocks. A macroblock is a rectangular block of 16\*16 luminance pixels and corresponding chrominance pixels; see video part of ISO 11172.

**6.3.2.3 Pixel Aspect Ratio**

Within the Full Motion system no constraints on the pixel aspect ratio are applied. Applications should take into account however that Full Motion decoders do not compensate for aspect ratio distortion. For MPEG SP streams it is therefore recommended to apply the same aspect ratio values as recommended for MPEG Video streams; see Chapter IX.4.3.2.3.



---

**IX.6 MPEG Still Picture**

---

**6.3.2.4 Picture Rate**

The Picture Rate indicates the number of pictures coded per second. For a still picture the Picture Rate is a parameter with a limited significance. The Picture Rate however relates the number of bits used to code the still picture to the Bitrate parameter. See the following section. The available Picture Rate options are 23.976 Hz, 24 Hz, 25 Hz, 29.97 Hz and 30 Hz, the same as allowed for "Constrained Parameter Streams".

**6.3.2.5 Bitrate**

Within the Full Motion System the Bitrate of an MPEG SP stream is bounded by the maximum size of the VBV buffer (see the following section) and the Picture Rate. The following relation between these three parameters shall apply for each MPEG SP stream:

$$\text{Bitrate} \leq \text{VBV(max)} * \text{Picture Rate},$$

- where
- Bitrate is the value encoded for the Bitrate field;
  - VBV(max) is the maximum value which may be encoded for the VBV Buffer Size field in MPEG SP streams;
  - Picture Rate is the value encoded for the Picture Rate field.

The value to be coded in the Bitrate field is bounded from below by the number of bytes used to code the still picture and the Picture rate. The following relation is to be applied for each MPEG SP stream:

$$\text{Bitrate} \geq N * \text{Picture Rate},$$

- where
- Bitrate is the value encoded for the Bitrate field;
  - N is the number of bits of the still picture video sequence, including the sequence header code and the sequence end code;
  - Picture Rate is the value encoded for the Picture Rate field.

**6.3.2.6 VBV Buffer Size**

The size of the VBV Buffer in each MPEG SP stream shall be smaller than or equal to 46 KByte (= 47,104 Bytes). The maximum number of bits to code one still picture is therefore 46 KByte.

---

**IX.6 MPEG Still Picture**

---

**6.3.2.7 Picture Coding Types**

The Pictures Coding Type identifies whether a picture is an intra-coded picture (I type), predictive-coded picture (P type), bidirectionally-predictive-coded picture (B type), or intra-coded with only DC coefficients (D type). In MPEG SP streams only one picture type is allowed: I-type pictures. The three other types are forbidden.

**6.3.2.8 User Data**

User data is application specific data. MPEG SP streams may contain User Data, but Full Motion decoders will ignore such data.

**6.3.2.9 Extension Data**

Extension Data may be defined by ISO in future. MPEG Video streams may contain Extension Data, but Full Motion decoders ignore such data.

**6.3.3 Access to MPEG SP Streams**

Applications have random access to MPEG SP streams at each picture. Before decoding, the application can store at the decoder the horizontal size and vertical size parameters from the sequence header. After decoding, the application can read these parameters from the decoder.

**6.3.4 Length of an MPEG SP Stream**

The minimum length of an MPEG SP stream is one still picture. The maximum length is only limited by the available disc capacity.

---

**IX.6 MPEG Still Picture**

---

**6.4 Presentation of MPEG Still Picture****6.4.1 General**

After decoding a (one picture) video sequence from an MPEG SP stream, a reconstructed still picture is available. Depending on the applied MPEG SP Decoder in the CD-I System, one to sixteen reconstructed still pictures can be stored in the MPEG SP Decoder. Default one still picture can be stored. Each stored still picture may have the maximum size specified in Chapter IX.6.3.2.2. One of the reconstructed still pictures or a part thereof can be displayed in the full motion video plane. The application selects which stored still picture is displayed.

**6.4.2 Full Motion Video Plane**

A reconstructed MPEG Still Picture or a part thereof can be displayed in the full motion video plane. During display of reconstructed MPEG Still Pictures, the full motion video plane is in normal horizontal resolution, i.e. with 360 or 384 pixels on a line, and in double vertical resolution, i.e. with 480 or 560 lines.

The reconstructed still picture or the part thereof which is displayed may be smaller than the spatial size of the full motion video plane. In areas of the full motion video plane which are not covered by video from the reconstructed still picture, a background color is displayed. The application can program one background color on RGB level for the whole plane. For each R, G and B component the integer values 0 to 255 are available; to applications it is recommended however to use only values between 16 and 235; black level is at 16 and nominal white peak level is at 235.

The full motion video plane is fully synchronized with the other CD-I planes. The width and the height of the full motion video plane correspond to the width and height of the other CD-I planes in terms of normal resolution. The width of all planes is 384 or 360 pixels of normal horizontal resolution and the height of all planes is 240 or 280 lines. See also Appendix V.2.2.1. The field rate at the full motion video plane can be 50 Hz or 60 Hz, following exactly the vertical synchronization of the other CD-I planes. The full motion video plane as well as the other CD-I planes shall be in the interlace mode. The MPEG SP decoder is informed by the application on the required width of the full motion video plane.

---

**IX.6 MPEG Still Picture**

---

The full motion video plane is available as External Video in the Video Decoder model. See Chapter V.2.6, Chapter V.4.1 and figure V.23. The full motion video plane replaces the backdrop plane, if the external video is enabled by setting the EV bit (External Video Enable) in the Select Image Coding Methods Instruction. See Chapter V.4.6.1 and figure V.47.

**6.4.3 Mixing with CD-I Video**

The mixing between the full motion video plane and the other CD-I planes is the same for MPEG Video and MPEG SP. See Chapter IX.4.4.2.

**6.4.4 Display Control Functions****6.4.4.1 Display Window****6.4.4.1.1 General**

After decoding a video sequence from an MPEG SP stream, a reconstructed still picture is stored at the decoder. Depending on the applied MPEG SP Decoder in the CD-I System, one to sixteen reconstructed still pictures can be stored. The application selects one of the stored still picture to be displayed in the full motion video plane. The part thereof to be displayed is selected by means of the display window. The position of this window in the full motion video plane is controlled by the application.

**6.4.4.1.2 Size of Display Window**

The display window has a rectangular shape, and cannot be larger than the decoded still picture. As a consequence, the width and height of the window should be smaller than or equal to the width and height of the decoded picture. Within these conditions, the width of the window is an integer number of pixels and the height is an integer number of lines.

The window should not exceed the border of the decoded picture. If the position of the window does not allow the window to remain within the picture, the MPEG Video Decoder clips the display window at the edge of the decoded picture. The application is informed about such clipping. The MPEG Video Decoder will use the clipped values of width and height of the display window until the width and height values are updated by the application.

---

**IX.6 MPEG Still Picture**

---

The size of the display window can be changed every coded picture period, as defined by the value encoded in the Picture Rate field in the MPEG SP stream. A modified window position will be accepted for display in the full motion video plane in the vertical blanking period following a vertical sync.

**6.4.4.1.3 Position within Decoded Still Picture**

The display window can be positioned within the decoded still picture by defining the address of the left upper corner of the window. This address can be any pixel/line position within the decoded still picture.

Each coded picture period, as defined by the value encoded in the Picture Rate field in the MPEG SP stream, the display window can be moved to an arbitrary position within the decoded still picture. A modified position within the decoded still picture will be accepted in the vertical blanking period following a vertical sync.

**6.4.4.1.4 Scrolling, Opening and Closing**

For scrolling the position of the display window within the decoded still picture can be modified each display field period. To open and close the window slowly, the size of the window can be modified each display field period. The total modification per display field period of size and position within the decoded still picture is constrained. These constraints are the same for MPEG Still Pictures and for MPEG Video; see Chapter IX.4.4.3.1.4.

**6.4.4.1.5 Blanking**

By the application, the display window can be blanked. In that case the contents of the display window will become 'black': R,G,B = 16,16,16. The blanking of the display window will remain active until the blanking is switched off by the application. After removing the blanking, the selected part of the decoded picture will be available again in the display window. In case no still picture is played back, the blanking is switched off at the first vertical sync after issuing the command. When a video sequence is played back, the blanking is switched off at the vertical sync immediately prior to the start of presentation of the decoded still picture.

---

**IX.6 MPEG Still Picture**

---

**6.4.4.1.6 Position within Full Motion Video Plane**

Within the full motion video plane one display window can be displayed simultaneously. The position of the display window within the full motion video plane can be controlled by the application by defining the position of the left upper corner of the window within the plane. The window can be positioned with an horizontal accuracy of one pixel of normal horizontal resolution and a vertical accuracy of one line of double vertical resolution (480 or 560 lines).

Each coded picture period, as defined by the value encoded in the Picture Rate field in the MPEG SP stream, the display window can be moved to an arbitrary position within the full motion video plane.

In order to make scrolling possible, the position of the display window within the full motion video plane can also be modified each display period. The displacement per display period is constrained: the displacement is to be in the range between +16 and -16 pixels or lines.

In all cases of a modified window position, the new position will be accepted for display in the full motion video plane during the vertical blanking period following a vertical sync.

If the window is positioned such that the window exceeds the border of the plane, the part of the window which exceeds the border of the plane is not displayed.

**6.4.4.2 Multiple Still Pictures**

Default one reconstructed still picture can be stored. Depending on the applied MPEG SP Decoder in a CD-I System, more than one reconstructed still pictures can be stored, with a maximum of sixteen. Each reconstructed still picture is stored in a page. The application selects which page is to be displayed in the full motion video plane. If the application selects a page, available in the decoder, in which no still picture is stored yet, the decoder will blank the display window.

---

**IX.6 MPEG Still Picture**

---

**6.4.5 Playback Control Functions****6.4.5.1 General**

MPEG SP streams can be played back in one mode: play still picture. A decoder can play simultaneously only one MPEG SP stream. During play back of an MPEG SP stream, the application shall not switch to another ISO 11172 stream. Depending on the applied MPEG SP Decoder in the CD-I System, one to sixteen reconstructed still pictures of maximum picture size (see Chapter IX.6.3.2.2.) can be stored in the MPEG SP Decoder. Which stored still picture is displayed is decided by the application.

The application can switch between different MPEG SP streams within an ISO 11172 stream. After switching to a new stream, the MPEG SP decoder will not accept data from the previous stream anymore. The first picture decoded from the new MPEG SP stream is the first still picture of which the first byte of the picture start code is encountered in that stream. The decoding of the new stream has the highest priority; the MPEG SP decoder may decide not to finalize completely the decoding of a still picture from the previous stream. The application may decide to display the last still picture from the previous stream until the first still picture from the new stream is available.

The functionality described in the following sections is provided for playback from disc as well as from memory, unless stated otherwise.

**6.4.5.2 Access Address in MPEG SP stream**

The application defines which still picture is decoded. In case of a play from disc, the access position within an MPEG SP stream can be addressed by means of an MPEG File Pointer and an MPEG SP Pointer. For byte indices in MPEG Video sectors see e.g. Chapter II.4.8.1.

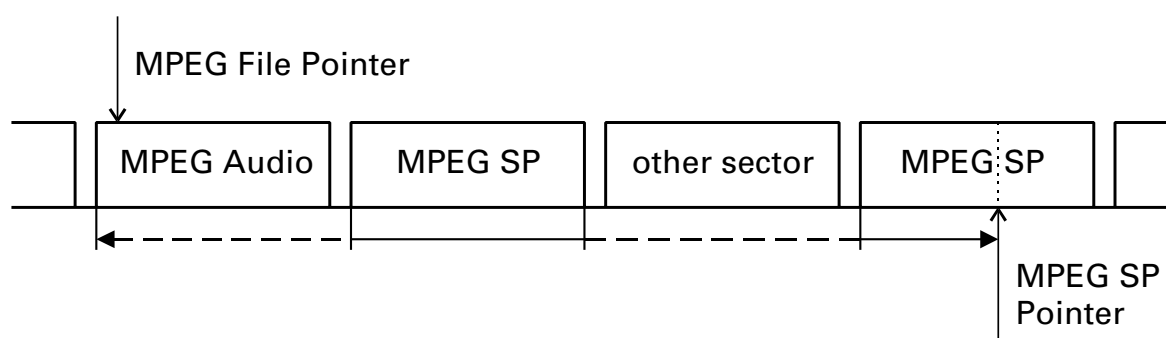
The MPEG File Pointer addresses the first sector to be retrieved from disc. This sector may be related to other playback requirements, e.g. of MPEG Audio. The sector addressed by the MPEG File Pointer may therefore be any sector from the file with the MPEG SP stream to be played. Applications should take into account that file pointers use a sector size of 2048 bytes to calculate a position within a file; see Chapter VII.2.2.3.2, the description of a function to issue a seek command to the disc drive (SS\_Seek).

## IX.6 MPEG Still Picture

The MPEG SP Pointer addresses the first byte to be accessed in the ISO 11172 stream. The MPEG SP Pointer shall address the first byte of a pack start code, or the first byte of a sequence header code, group start code or picture start code. See system part and video part of ISO 11172.

In case of an access from disc, the MPEG SP Pointer addresses the first byte of the access by indicating the index of that byte within the ISO 11172 stream from the selected CD-I channel. Note that one CD-I channel may contain sub-subsequent ISO 11172 streams with MPEG SP data. See Chapter IX.6.2.3. The first byte of the ISO 11172 stream retrieved from disc in the selected CD-I channel has index zero. See also the example in figure IX.6.3.

Figure IX.6.3 Example MPEG File Pointer and MPEG SP Pointer



MPEG File Pointer points to MPEG Audio sector. Data retrieval from disc will therefore start with this sector.

MPEG SP Pointer indicates the index of a byte in the ISO 11172 stream from the selected CD-I channel. If this Pointer has the value zero, the access will start at the pack header from the first MPEG SP sector retrieved from disc in the selected CD-I channel.

In this example the MPEG SP Pointer indicates the first byte of (e.g.) a sequence header code. In the example the MPEG SP Pointer equals 3592; therefore the first accessed byte of the MPEG SP stream is the 1269th byte ( $3592 - 2324 + 1 = 1269$ ) of the second MPEG SP sector retrieved from disc in the selected CD-I channel.

In case of an access from memory, only the MPEG SP Pointer is used. In such case, the MPEG SP Pointer indicates the index of a byte from the ISO 11172 stream with MPEG SP data stored in memory. The first byte of such ISO 11172 stream has index zero.



## IX.6 MPEG Still Picture

---

Decoding will start at the first still picture encountered in the MPEG SP stream after the byte referred to by the MPEG Video Pointer. A picture is encountered in the stream if the first byte of the picture start code is encountered. Parameters from a sequence header or from the group of pictures layer are read by the decoder before starting the decoding, if the first byte of a sequence header code or of a group start code from the accessed MPEG Video stream is read prior to the first byte of the picture start code. It is the responsibility of the application that all required sequence parameters are available at the decoder when the decoding of a still picture starts.

### 6.4.5.3 Play Still Picture

In the play still picture mode a still picture from an MPEG SP stream is decoded and subsequently stored in the decoder. Which still picture is to be played is indicated by the application by means of an access address. See Chapter IX.6.4.5.2. The application controls in which page a reconstructed still picture is stored. If in a page a still picture has been stored already, the stored picture will be overwritten by the played picture. The application selects which page, i.e. which decoded still picture, is displayed. See Chapter IX.6.4.4.2.

### 6.4.6 MPEG SP Memory Maps

The Full Motion System supports play of MPEG SP maps from memory. Each ISO 11172 stream played from memory and each MPEG SP stream multiplexed in that ISO 11172 stream shall meet the specifications from the ISO 11172 standard. These streams shall further meet the constraints for MPEG parameters as specified in Chapter IX.6.3. The map in memory of an ISO 11172 stream, played from memory, shall commence with the first byte of a pack start code.

Play back of MPEG SP from disc and play from memory are mutually exclusive functions. Play back of MPEG SP streams is possible from disc or from system memory with the same playback functionality. During an MPEG SP play from memory, the maximum bitrate for ISO 11172 streams and for MPEG Video streams is bounded by constraints specified in Chapter IX.6.3.1 and IX.6.3.2.5.

A play of an MPEG SP map starts with the byte in the ISO 11172 stream indicated by the MPEG SP Pointer. See Chapter IX.4.4.4.2.

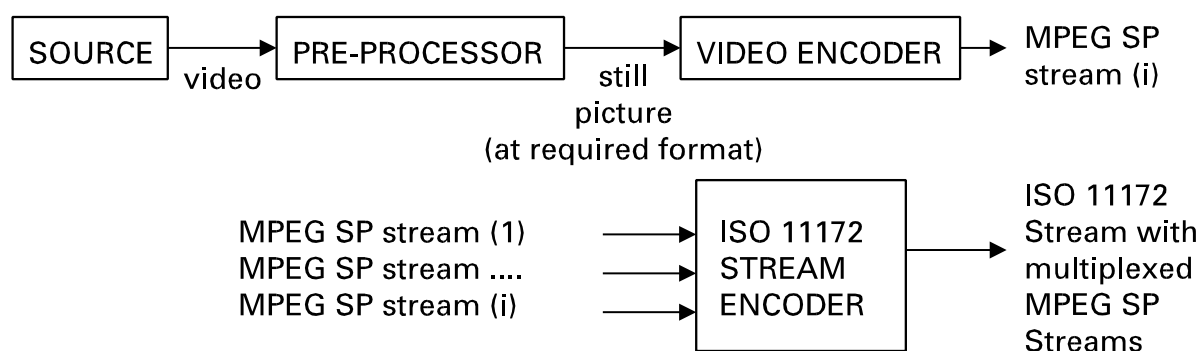
## IX.6 MPEG Still Picture

## 6.5 MPEG SP Data Encoding

## 6.5.1 MPEG SP Encoder Model

The MPEG SP encoder model is depicted in figure IX.6.4. The source, e.g. a video tape recorder, delivers a video signal to the pre-processor; output of the pre-processor is the still picture to be coded. A pre-processed still picture has the format which is used for encoding. The pre-processed still picture is input to the video encoder. At the output of the video encoder, a coded MPEG SP stream is available. Finally one or more MPEG SP streams are input to the ISO 11172 stream encoder, which produces an ISO 11172 stream at the format required for storage in the datafield of an MPEG SP sector.

Figure IX.6.4 MPEG SP Encoder Model



## 6.5.2 Pre-processor

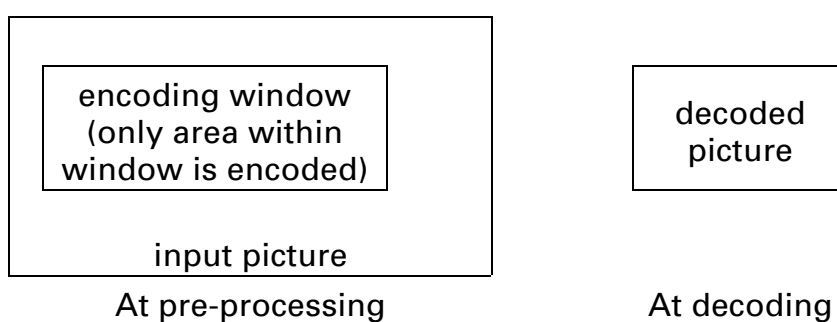
The MPEG Video Encoder requires as input a still picture in the format at which the still picture is to be encoded. Normally such format is not directly delivered by the source. On the video output from the source therefore pre-processing is applied. In case of an analog input signal, A to D conversion is included in the pre-processing, as encoding requires a digital input format. The techniques used for pre-processing are at the discretion of the implementation.

One of the pre-processing functions is to grab the still picture. Grabbing may be depend on whether the source material originates from film or from video camera. In case of film, the still picture can be derived from a film image which is recovered from the video data.

## IX.6 MPEG Still Picture

Further functions of the Pre-processor may be filtering and windowing. Filtering may be needed to reduce spatial resolution, e.g. before sub-sampling. If only a part of a grabbed still picture is to be coded, the area which is to be coded can be defined by means of an encoding window. The size of the window is limited by the constraints for the size of the coded picture. See Chapter IX.6.3.2.2. The encoding window can be positioned within a grabbed input picture, independent of the spatial resolution of the grabbed picture. For an example of an encoding window within an input picture see figure IX.6.5.

Figure IX.6.5 **Example of encoding window within input picture**



The area within the encoding window will be coded; at decoding the part of the input picture which is coded will be reconstructed.

### 6.5.3 Video Encoder

The Video Encoder produces an MPEG SP stream from the still pictures provided at the input. Within the flexibility provided by ISO 11172 and within the constraints described in Chapter IX.6.3.2, it is at the discretion of the implementation which encoding techniques are applied.

Parameters used during encoding depend on the application. Within the still picture to be coded slices are to be defined (see video part of ISO 11172). The number of bits for coding a picture is either specified by the application before encoding or is defined at encoding, as the result of the applied encoding strategy.

**6.5.4 ISO 11172 Stream Encoder**

The ISO 11172 stream encoder multiplexes one or more MPEG SP streams into one 11172 stream. The format of the produced ISO 11172 stream meets the constraints described in Chapter IX.6.3.1. The multiplex depends on which sectors are allocated as MPEG SP sectors. The MPEG SP sector allocation is therefore either provided before multiplexing or is specified by the ISO 11172 stream encoder. The applied multiplex technique is at the discretion of the implementation.

---

**IX.6 MPEG Still Picture**

---

**6.6 MPEG SP Decoding****6.6.1 MPEG SP Decoder****6.6.1.1 MPEG SP Decoder Model**

For MPEG SP decoding the same model is applied as for MPEG Video decoding. See figure IX.4.9. The MPEG SP Decoder Model requires in Chapter IX.4.6.1.1 replacement of the text "MPEG Video sectors" by "MPEG SP sectors" as well as replacement of the text "MPEG Video stream" by "MPEG SP stream". The Model consists of three modules: the Transfer Buffer, the MPEG Video Decoding Unit and the MPEG Video Buffer. The requirements for the modules from MPEG SP streams are slightly different than those from MPEG Video streams.

**6.6.1.2 Transfer Buffer**

In the Transfer Buffer data are stored from the ISO 11172 stream in MPEG SP sectors from the selected CD-I channel. The data are stored in the Transfer Buffer by the CD-I Sector Processor or the Microprocessor Unit. The data are read by the MPEG Video Decoding Unit.

In case of a play from disc, the Transfer Buffer is a chain of at least 2 circularly linked PCL Buffers, where each buffer has room to store 1 MPEG SP sector (2324 bytes); see Chapter VII.2, Play Control List (PCL) Format.

The provision of the Transfer Buffer is the responsibility of the application.

**6.6.1.3 MPEG Video Decoding Unit**

The MPEG Video Decoding Unit reconstructs a still picture from an MPEG SP stream. Input to the MPEG Video Decoding Unit is an ISO 11172 stream. A still picture is reconstructed from one (still picture) video sequence from an MPEG SP stream in the multiplex of the ISO 11172 stream. From which MPEG SP stream a still picture is reconstructed is controlled from the application by the Microprocessor Unit. After reconstruction, the still picture is displayed in the full motion video plane at the output of the MPEG Video Decoding Unit. Within the MPEG parameter constraints described in Chapter IX.6.3, the MPEG Video Decoding Unit meets the specifications for an MPEG Video decoder from the ISO 11172 standard.

## IX.6 MPEG Still Picture

---

### 6.6.1.4 MPEG Video Buffer

The MPEG Video Buffer is used by the MPEG Video Decoding Unit for storage purposes. A decoded still picture is stored in the MPEG Video Buffer and displayed at the frame rate of the full motion video plane. The size of the MPEG Video Buffer depends on the implementation; see Chapter IX.4.6.1.4.

### 6.6.2 Audio/Video Synchronization

A play of an MPEG SP stream can be synchronized with a play of MPEG Audio, ADPCM Audio, CD-DA or Base Case Video. Both the Synchronization Model and the requirements as described in Chapter IX.4.6.2 for MPEG Video do also apply for an MPEG SP play. Condition is that the MPEG Video Decoder meets the requirements for an MPEG SP Decoder, specified in Chapter IX.6.6.1.

### 6.6.3 Synchronization between Events

To synchronize events in an application with events in an MPEG SP play, the application is, upon request, informed about the MPEG SP event that the display of a decoded still picture starts. A still picture starts being displayed when at the output of the MPEG SP Decoder the first field derived from that still picture is displayed. The event is generated before the active video starts at the output of the MPEG SP decoder of that first field, but after the vertical V-sync preceding such active video.

### 6.6.4 Device Status Descriptor MPEG SP Decoder

The Device Status Descriptor for MPEG Video Decoders in a CD-I System, referred to in Chapter IX.4.6.4, indicates also how many reconstructed still picture can be stored in the MPEG SP Decoder. In an MPEG SP Decoder no more than sixteen reconstructed still pictures can be stored; default one still picture can be stored.

---

**IX.6 MPEG Still Picture**

---

**6.7 MPEG SP Data Re-arrangements**

The application may re-arrange MPEG SP data. Each MPEG SP stream in an ISO 11172 stream as well as each ISO 11172 stream resulting from such re-arrangements shall meet the specifications from the ISO 11172 standard. These streams shall further meet the constraints for MPEG parameters as specified in Chapter IX.6.3.

**6.8 Error Performance**

Two error types in MPEG SP data are distinguished for error performance: flagged and un-flagged errors. The CD-I system may indicate flagged errors by means of an error flag. Un-flagged errors are errors which are not recognized; the decoder treats data with un-flagged errors as error-free data.

In case of a flagged error, the MPEG SP Decoder may abort the decoding process. In case of an abort the application shall be informed about such event.

In case of un-flagged errors, the decoding proceeds with erroneous data; as a consequence, the decoding process may reach an undefined state and the video output of the decoder may be corrupted.

This page is intentionally left blank



---

**IX.7 Memory Extension**

---

**IX.7 Memory Extension****7.1 General**

The MPEG Video Decoding Model consists of a Transfer Buffer, an MPEG Video Decoding Unit and an MPEG Video Buffer; see Chapter IX.4.6.1.1 and figure IX.4.9. Characteristics of this model in case of an MPEG Video play are described in Chapter IX.4.6.1 and in case of an MPEG SP play in Chapter IX.6.6.1. In case neither an MPEG Video stream nor an MPEG SP stream is played, the MPEG Video Buffer from the MPEG Video Decoding Model can be used by the application to extend memory. In that case the MPEG Video Decoding Unit serves as controller of the MPEG Video Buffer, such that the MPEG Video Buffer is transparently accessible from the Microprocessor Unit as a memory block.

**7.2 Memory size**

The minimum size of the MPEG Video Buffer equals 512 KByte. The maximum size of the MPEG Video Buffer is not constrained. The size of the MPEG Video Buffer in a CD-I System is indicated in the Device Status Descriptor of the MPEG Video Decoder. See Chapter IX.4.6.4. The size is indicated as  $B \cdot 1024 \cdot 1024$  bits, where B is an integer number with a minimum value of 4. The maximum value of B is not constrained.

Applications should take into account that 32 KByte of the memory bank is not be available to the application. The minimum memory size available to applications is therefore 480 KByte.

**7.3 Memory access**

In case of applying the MPEG Video Buffer for memory extension, the MPEG Video Buffer can be accessed directly from the Microprocessor Unit as a memory bank of  $B \cdot 64 \cdot 1024 \cdot 16$  bits, where B is an integer number with a minimum value of 4. The maximum value of B is not constrained. For transparency towards applications, byte access as well as word access are supported.

IX.7 Memory Extension

---

7.4 **Memory color**

For memory extension, the memory color of the MPEG Video Buffer is defined to be \$90. See also Appendix VII, OS9 technical overview, the Kernel, Colored Memory, Parameter Memory Type, page 2-11 and 2-12.

IX.8 CD-RTOS for Full Motion

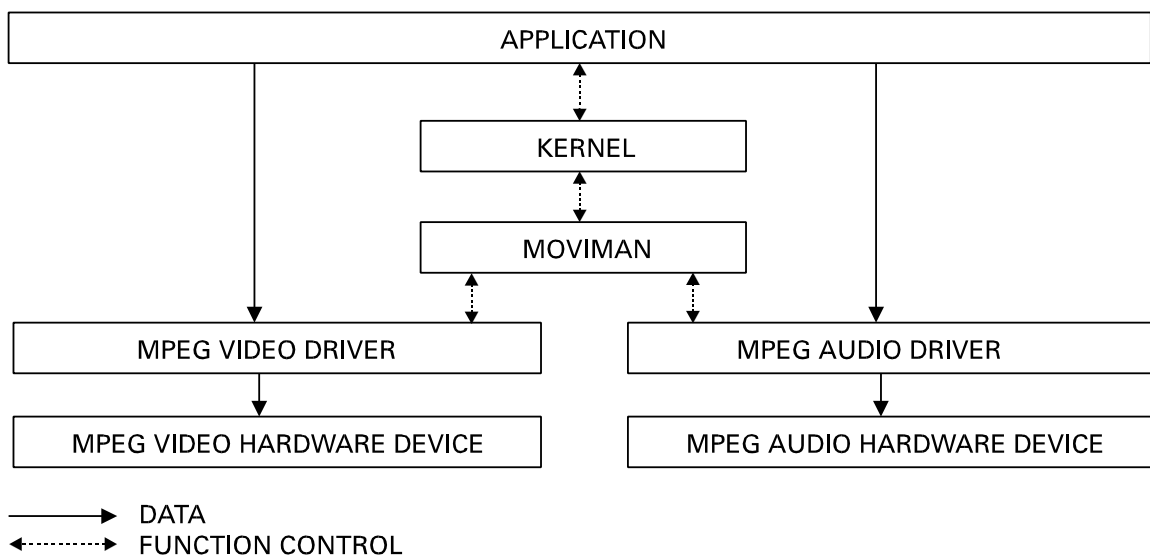
IX.8 CD-RTOS for Full Motion

8.1 General

The Full Motion system extends a CD-I player with an MPEG Video decoder and an MPEG Audio decoder. The MPEG Video decoder is used to decode MPEG Video streams or, in another mode, MPEG Still Picture streams. The MPEG Audio decoder is used to decode MPEG Audio streams. The MPEG Video decoder as well as the MPEG Audio decoder consist of a hardware device and a device driver. Both drivers are implementation dependent. The application controls decoding by communication with the drivers. The application communicates with the drivers via the Full Motion Manager (moviman), see figure IX.8.1. The Full Motion Manager is part of CD-RTOS. For the Full Motion system, the CD-RTOS specification is therefore extended with the Full Motion Manager, with function calls between the application and the Full Motion Manager and with function calls between the Full Motion Manager and both device drivers.

The application program issues requests for system services through the CD-RTOS kernel. The Full Motion extension of CD-I has no impact on the CD-RTOS kernel. For detailed information on the CD-RTOS kernel, see Chapter VII.1 and Appendix A VII.1, Chapters 1-4.

Figure IX.8.1 Full Motion Data Flow in CD-RTOS



## 8.2 Full Motion Manager (moviman)

### 8.2.1 Introduction

The moviman is a file manager module that supports requests to the MPEG Video driver and the MPEG Audio driver. The moviman provides the following functional capabilities:

- a) to play MPEG Video streams and MPEG Still Picture streams;
- b) to play MPEG Audio streams;
- c) support to synchronize plays of MPEG Video and MPEG Audio;
- d) to present decoded pictures from MPEG Video streams or MPEG Still Picture streams in the full motion video plane at the output of the MPEG Video decoder;
- e) to present decoded MPEG Audio signals into the Left and Right audio signals at the output of the MPEG Audio decoder.

Note that the User Communication Manager (UCM) supports switching between the full motion video plane and the base case planes.

### 8.2.2 Conventions in Notation

Function calls are described in the same manner as the functions in Chapter VII. Register usage is as uniform as possible for all calls. In practice this means that on:

input: register d0.w is always the path number, register d1.w is always the set/getstat function code, register d2.w (where applicable) is the mapID and address parameters are in registers a0, a1 etc.

output: depending on the function, d0, d1 etc contain data type values, and a0, a1 etc contain address type values. Also on output, in the case of an error, the carry flag will be set and the appropriate error code will be in d1.w

The following abbreviations are used:

MVM	MPEG Video Map
ID	Identifier

All coordinate and size related parameters are according to the UCM conventions.

### 8.2.3 MPEG Video Functions

To control MPEG Video decoding, the application calls the Full Motion Manager. The MPEG Video functions available to the application are described in this section. With each function, its purpose, its constraints and its register usage (for input and for output) are described. Any data structures, used for communication between application and moviman, are described also.

#### 8.2.3.1 The Setstat and Getstat Functions

The setstat and getstat functions are, in alphabetical order, described in this section.

**8.2.3.1.1 MV\_Abort - Abort the current MPEG Video play**

This function aborts the play that is currently being executed. If no play is active, an abort error will be returned. The last displayed picture will remain visible.

This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which started the play.

If this play was running in **synchronized** mode with an audio play, then this call will end the synchronized mode. The audio playback will continue in non-synchronized mode.

Input:           d0.w = path number  
                  d1.w = MV\_Abort setstat function code

Output:           None

Error output:    cc     = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$Abort, E\$Permit

IX.8 CD-RTOS for Full Motion

---

**8.2.3.1.2 MV\_BColor - Set the border colour**

This function sets the border colour. If the specified MVM is currently active, the parameters are copied into the MVM descriptor and the parameters are activated immediately. If the MVM is not active, the parameters are copied into the MVM descriptor only.

This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which created the descriptor.

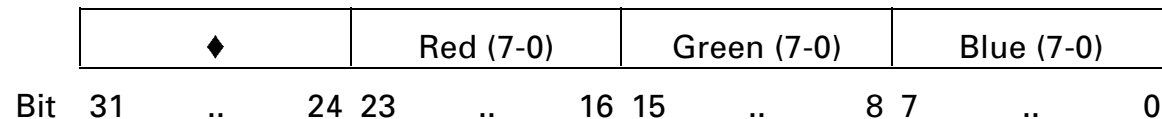
Input:           d0.w = path number  
                   d1.w = MV\_BColor setstat function code  
                   d2.w = MapID  
                   d3.l = color value

Output:           None

Error output:    cc    = carry bit set  
                   d1.w = error code

Possible errors: E\$BPNuM, E\$UnID, E\$Permit, E\$IIPrm

Figure IX.8.2           **The format of the color value parameter**



As specified in the CD-I standard, for each component, black level is at 16 and nominal peak (white) level is at 235.

### 8.2.3.1.3 **MV\_ChSpeed - Change the MPEG Video playback speed**

This function changes the speed of an active play. When changing to scan mode or single picture forward mode, the system will not restart decoding anymore. Any next picture must be requested with an MV\_Next call. If no MapID is active or when this call is made for a map of type Still or when the currently active play is frozen, a bad mode error is returned.

This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which started the play.

If, during **synchronized** mode, the application changes from normal speed to non-normal speed, the decoder will stay in synchronized mode and will mute the audio. If a change to normal speed takes place, audio will be demuted again and synchronization will be restored.

The format of the speed parameter is defined in Chapter IX.8.2.3.3.

The contents of the PCL parameter and the Offset to start parameter are ignored when:

- the new speed mode is scan mode (the call MV\_Next will provide the required values for d3.l and a1).
- the old speed is NOT scan mode. The system will continue reading from the current position.

#### **Playing from CD:**

If a play is already active, the PCL pointer **may** be passed to make the system fluently change to another chain of PCL buffers.

If the application requests for such a change, the system will link the new chain of PCL's to the first PCL structure of which the buffer is emptied. If the system is already changing to another chain of PCL buffers, this parameter is ignored which may result in using a chain of PCL's unfit for the new speed mode. When the system does no longer use the 'old' PCL structure, it will restore the link and the application will be informed (by the CNP bit in the status block or by the CNP signal if set with MV\_Trigger).

The Offset to start parameter must contain a byte offset (relative to the first incoming byte) to the first byte of a pack start code, or the first byte of a sequence header code, group start code or picture start code (see MPEG Video Pointer, Chapter IX.4.4.4.2).



**Consequences on changing the speed when playing from CD:**

- A change to normal speed can be done in two ways using the call
  - a) MV\_ChSpeed.  
Speed change is fluent, but synchronization between audio and video is only restored after a while, depending on the amount of data in the chain of PCL-buffers and the MPEG decoder.
  - b) MV\_Play.  
The speed change is not fluent, but audio and video are immediately synchronized.
- When changing to slow motion, the application must be sure to have enough data in the PCL-buffers to decode while pausing and continuing the CD-player.
- When changing from slow motion forward mode or single picture forward mode to normal forward mode, the system will not change to normal speed until a new sector from disc is detected or when a recommended time out period of 4 seconds has elapsed. After this speed change, the MPEG decoder must get rid of superfluous data by skipping pictures or frames.
- If the system is in scan mode and an MV\_Next call is issued, the MV\_ChSpeed call will return the E\$DevBsy error from the moment that data from disc is copied until the picture is displayed unless the new speed is scan speed.

**Playing from host:**

The PCL pointer parameter is not used. The Offset to start parameter must contain a byte offset (relative to the start of the map) to the first byte of a pack start code, or the first byte of a sequence header code, group start code or picture start code (see MPEG Video Pointer, Chapter IX.4.4.4.2).

Input:           d0.w = path number  
                   d1.w = MV\_Speed setstat function code  
                   d3.l = Offset to start  
                   d5.l = Speed parameter  
                   (a1) = PCL-pointer (CD only)

Output:           None

Error output:   cc    = carry bit set  
                   d1.w = error code

Possible errors: E\$BPNum, E\$BMode, E\$IIIPrm, E\$DevBsy, E\$Permit

**8.2.3.1.4 MV\_Close - Free the MVM descriptor in use**

This function aborts any ongoing actions on the specified MVM and frees the used MVM descriptor. The last displayed picture will remain visible.

If another process has a play request queued, using this MVM descriptor, then that process will be activated to return an E\$UnID error.

This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which created the descriptor.

Input:           d0.w = path number  
                  d1.w = MV\_Close setstat function code  
                  d2.w = MapID

Output:           None

Error output:    cc    = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$UnID, E\$Permit

**8.2.3.1.5 MV\_Conceal - Replace errors in MPEG Video datablock**

This function, for host only, replaces suspected data in an MPEG datablock. If the map to which the specified data belongs is currently in use (= data retrieved from), data integrity is not guaranteed for the complete map.

Error concealment is automatically done during a Full Motion video play that makes use of a PCL provided that the PCL also includes the error structure.

Input:           d0.w = path number  
                  d1.w = MV\_Conceal setstat function code  
                  d3.l = size of the datablock to correct  
                  (a0) = pointer to error structure  
                  (a1) = pointer to the data

Output:           None

Error output:    cc    = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$IIPrm

The error structure is identical to the error structure used in the PCL-structure for the SS\_Play command. See Chapter VII.2.2.3.2.

**8.2.3.1.6 MV\_Continue - Continue the paused or frozen MPEG Video play**

This function continues a paused or frozen MPEG Video play. If the play is not in paused or frozen state or if no MPEG Video play is active, a bad mode error will be returned.

This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which paused the play.

If the MPEG video play is active in synchronized mode with MPEG audio, and if this synchronized play is paused or frozen, this function will cause both the video and the audio to continue.

If the play was **frozen**, decoding and displaying starts according the filter parameter. During the time the play is frozen, data retrieval continues, so this has not to be restarted.

If the play was **paused**, the data retrieval from CD must be restarted after this call. In case of normal speed play, data available in the internal buffer(s) will be decoded when new data from disc is received or when a recommended time out period of 4 seconds has elapsed.

Data retrieval from 'host' is restarted automatically by the decoder. The application does not have to do any additional (seek) actions.

The filter parameter is ignored when the player is continued from a paused state.

Input:           d0.w = path number  
                  d1.w = MV\_Continue setstat function code  
                  d2.b = filter parameter

Output:           None

Error output:    cc    = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$BMode, E\$Permit, E\$IIPrm

IX.8 CD-RTOS for Full Motion

---

The values of the filter parameter:

- 0: Resume decoding as soon as possible, provided that the sequence parameters are known by the MPEG decoder (see Chapter IX.4.3.4)
  - 1: Resume decoding at the next group of pictures, provided that the sequence parameters are known by the MPEG decoder (see Chapter IX.4.3.4)
  - 2: Resume decoding at the next sequence header
- Other values are reserved.

IX.8 CD-RTOS for Full Motion

---

8.2.3.1.7 **MV\_Create - Reserve an MPEG Video descriptor**

This call reserves and initializes an MVM descriptor. The format of the MVM descriptor, the initial values and usage of its fields can be found in Chapter IX.8.2.3.3

If the selected type is 'still', the MV\_Play call returns an E\$IIIPrm error when the page number is out of bounds. Page zero is the first selectable page.

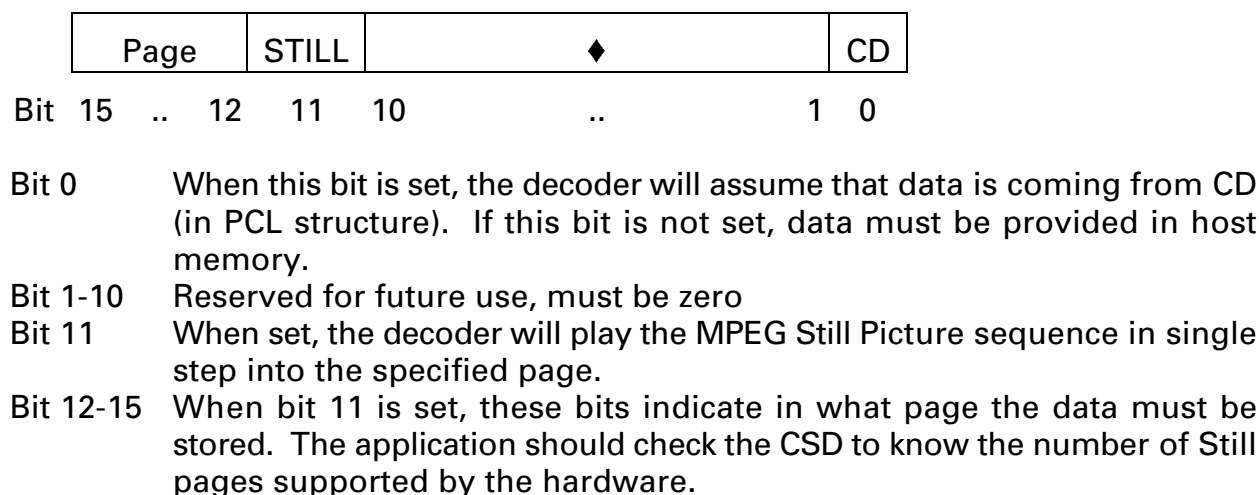
Input:           d0.w = path number  
                   d1.w = MV\_Create getstat function code  
                   d2.w = MVM type

Output:           d0.w = MapID

Error output:   cc    = carry bit set  
                   d1.w = error code

Possible errors: E\$BPNum, E\$IIIPrm, E\$MemFul, E\$NoRam

Figure IX.8.3           **The format of the type parameter**



**8.2.3.1.8 MV\_Freeze - Freeze the current picture (CD only)**

This function causes the output of a normal speed MPEG Video play to be frozen, while the data retrieval continues. Output will be resumed after issuing an MV\_Continue call. If no normal speed play is active, or if the play is already frozen or paused, a bad mode error will be returned.

This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which started the play.

Input:           d0.w = path number  
                  d1.w = MV\_Freeze setstat function code

Output:           None

Error output:    cc    = carry bit set to one  
                  d1.w = error code

Possible errors: E\$BPNum, E\$BMode, E\$Permit

**8.2.3.1.9 MV\_Hide - Disable the window output**

This function disables the output of the window on the next vertical retrace. The window will become black but decoding continues.

This is a privileged function. Permission is granted if the function is called by the super user, if no play is currently active or if the currently active play was started by the same user that calls this function.

Input:           d0.w = path number  
                  d1.w = MV\_Hide setstat function code

Output:           None

Error output:    cc    = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$Permit

**8.2.3.1.10 MV\_ImgSize - Preset picture sizes**

This function sets in the MVM descriptor the picture size and picture rate of the sequence to be played.

This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which created the descriptor.

The parameters are specific to the sequence and cannot be changed by an application during a play (the call will return an E\$BMode error when active). Changing the picture size (by a call or by the data stream) may clip the specified window.

The width must be less than or equal to 768, the height must be less than or equal to 576. Except for stills, the picture rate can be 23, 24, 25, 29 or 30 and:

$$((H/2 + 15)/16) * ((V/2 + 15)/16) \leq 396$$

$$((H/2 + 15)/16) * ((V/2 + 15)/16) * \text{picture rate} \leq (396 * 25)$$

Input:           d0.w = path number  
                   d1.w = MV\_ImgSize setstat function code  
                   d2.w = MapID  
                   d4.l = Size (W:H) in UCM coord. (not zero)  
                   d5.b = Picture rate (23..25 or 29..30)

Output:           None

Error output:   cc    = carry bit set  
                   d1.w = error code

Possible errors: E\$BPNum, E\$IIIPrm, E\$UnID, E\$Permit, E\$BMode



**8.2.3.1.11 MV\_Info - Return pointer to MVM descriptor**

MV\_Info returns a pointer to the MPEG Video descriptor. The application may not alter the contents of the MVM descriptor.

This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which created the descriptor.

Input:           d0.w = path number  
                  d1.w = MV\_Info getstat function code  
                  d2.w = MapID

Output:           (a0) = pointer to MPEG Video descriptor

Error output:    cc    = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNuM, E\$UnID, E\$Permit

**8.2.3.1.12 MV\_Jump - Compensate video streams' timing parameters on behalf of seamless jump**

The application is allowed, when playing from CD, to discontinue the current data stream and to start supplying data that comes from another position. To overcome a discontinuity in timing parameters, this function provides the application a means to adapt the new timing parameters in such a way that they fit seamlessly into the old stream. As soon as either a play is started or the decoder detects a discontinuity in the supplied stream, it will start correcting the timing parameters with a given delta value. This delta value is defined as  $SCR(j) - SCR(i)$  and is given in a resolution of 22.5 kHz.  $SCR(j)$  is the value of the first sector of the new data and  $SCR(i)$  is the value of the SCR of the sector that would have been supplied next if no jump was done. Initially the delta value is zero. After a play is aborted, the delta value will be reset to zero. When playing from host, the delta value will have no influence at all. Since a discontinuity in SCR is detected only if the difference in between SCR's is negative or more than 0.70 seconds, a positive delta value less than or equal to 15,750 is considered as an illegal parameter.

Input:           d0.w = path number  
                  d1.w = MV\_Jump setstat function code  
                  d3.l = Delta value (signed)

Output:           None

Error output:   (cc) = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$IIIPrm

**8.2.3.1.13 MV\_Loop - Repeat a number of pictures**

This function sets the loopback points for a play from memory.

This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which created the descriptor.

If this MVM is currently not active, parameter checking is not possible since a link to a map will exist only after an MV\_Play is issued (until then no mapsize is available). When the map is activated by MV\_Play, the loop variables will be checked.

This call is only allowed for MVM's of type Host. For other types, a bad mode error will be returned. If the MVM is in scan mode, setting the loopback variables will have no immediate effect. As soon as the speed is changed to normal, slow motion or single step mode, the loopback function will be executed.

The Starting loopback point offset contains an MPEG Video Pointer (see Chapter IX.4.4.4.2), provided that the sequence parameters are either in the datastream or set by the application (MV\_SelStrm or MV\_ImgSize), the Ending loopback point offset must be on the byte following the last byte of the last picture to be included in the loop area. It is the application's responsibility to provide the correct addresses. The addresses must be relative to the start of the map.

When the MVM is afterwards played (through MV\_Play), it will play the pictures from the start of the looparea until the end of the looparea is reached. The first picture to be displayed is the first Access picture encountered in the loop area. The Loopcount will be decremented and when it did not reach zero, the images between the looppoints will be displayed again until the Loopcount is zero.

IX.8 CD-RTOS for Full Motion

---

If the MVM is already being played when this call is made, the behavior of this function depends on the current position in the MVM. If the current position is before the Ending loopback point offset, then playing continues until the Ending loopback point offset is reached, at which time the Loopcount is decremented and the looping area is played again, if necessary. If the current position is already behind the Ending loopback point offset, a jump will take place to the Starting loopback point offset at the next picture change.

The application is responsible for setting the correct sequence parameters before the play is started if these are not in the data itself. Every time a jump is made to the Starting loopback point offset, the decoder will set the sequence parameters as they are available in the descriptor.

If the MV\_Loop call is applied to a play which is already in loopback mode, then the currently active loop area will be completed first (i.e. played until the end position of the active loop area). At that moment, the loop parameters of the most recent MV\_Loop call will become active. How the play continues depends on the current position (which is the end position of the 'old' loop area). See the sub-section before the previous one.

If the designated mapID is active and in synchronized mode with audio, then this MV\_Loop call will result in a E\$BMode error.

Input:           d0.w = Path number  
                   d1.w = MV\_Loop setstat function code  
                   d2.w = MapID  
                   d3.l = Starting loopback point offset (bytes)  
                   d4.l = Ending loopback point offset (bytes)  
                   d5.w = Loopcount

Output:           None

Error output:   cc    = carry bit set  
                   d1.w = error code

Possible errors: E\$BPNum, E\$UnID, E\$BMode, E\$Permit, E\$IIPrm

**8.2.3.1.14 MV\_Next - Display next picture**

This function must be used to step to the first or the next picture when in single picture forward mode (or playing still pictures) or to tell the MPEG Video decoder that it must expect new data to display the next entry point when in scanning mode. In this mode, the next picture will be made visible when the current scantime (parameter in MV\_Play and MV\_ChSpeed calls) has elapsed.

This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which started the play.

The parameter 'page number' is only used when the active mapID is of the type 'Still'. It must be given to select the page to decode the next picture in. If the value does not fit the number of pages available in hardware, an E\$IIIPrm is returned. The first page is identified as page zero.

When playing from host in scan mode, the Offset to start parameter (relative to the start of the map) must be passed.

When in single step mode, the Offset to start parameter is ignored.

When playing from CD in scan mode, this call indicates that the chain of PCL-structure was emptied by the application and that new data is to be expected. The decoder does not accept a new MV\_Next call (device busy) until the picture is displayed (a PIC-signal (PIC/GOP/SOS/LPD) is sent).

When playing in single step mode the next picture will be made visible as soon as possible (data available and decoded). The PCL pointer and offset parameter are ignored in this mode.

If no play is active, an E\$BMode error is returned.

IX.8 CD-RTOS for Full Motion

---

A description of the parameter Offset to start is given in the MV\_ChSpeed function description.

Input:           d0.w = Path number  
                  d1.w = MV\_Next setstat function code  
                  d2.b = Page number (Stills only)  
                  d3.l = Offset to start  
                  (a1) = PCL-pointer (CD only)

Output:           None

Error output:    cc    = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$BMode, E\$Permit, E\$DevBsy, E\$IIPrm

**8.2.3.1.15 MV\_Off - Turn off the display output**

This function disables the output of the display. The screen will become black but decoding continues.

This is a privileged function. Permission is granted if the function is called by the super user, if no play is currently active or if the currently active play was started by the same user that calls this function.

Input:           d0.w = path number  
                  d1.w = MV\_Off setstat function code

Output:           None

Error output:    cc    = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$Permit

**8.2.3.1.16 MV\_Org - Set origin offset**

MV\_Org sets the origin of the window relative to the top left corner of the full screen image. This facilitates the overlay of the graphics cursor. Default origin is the upper left corner of the full screen image, coordinates 0:0.

If the specified map is currently active, the new parameters will become effective on the next picture change. If the scroll parameter (d5.b) is not zero and the window move is over less than 16 lines, the window will be repositioned on the next vertical retrace.

This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which created the descriptor.

Input:           d0.w = path number  
                  d1.w = MV\_Org setstat code  
                  d2.w = MapID  
                  d3.l = origin (H:V) in UCM coordinates  
                  d5.b = scroll flag (not 0: scroll, 0: no scroll)

Output:           None

Error output:    cc    = carry bit set on error  
                  d1.w = error code

Possible errors: E\$BPNum, E\$UnID, E\$Permit



**8.2.3.1.17 MV\_Pause - Pause the current play**

This function pauses the MPEG Video play that is currently being executed. The current play must be either in 'normal' speed or in 'slow motion' speed, else a E\$BMode error will result. If no MPEG Video play is active at the time this function is called or if the active play is already paused or frozen, a bad mode error is returned. If the MPEG Video play is in synchronized mode with MPEG Audio, this function will pause both the video and the audio playback.

This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which started the play.

Input:           d0.w = path number  
                  d1.w = MV\_Pause setstat function code

Output:           None

Error output:    cc    = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNuM, E\$BMode, E\$Permit

**8.2.3.1.18 MV\_Play - Start an MPEG Video play**

This function starts to play the data belonging to the given MPEG Video mapID. The data may be coming from disc or from host memory. Play is an asynchronous operation, i.e. the application continues execution at the same time as the play is executed. To have the first picture displayed when the playback speed is scan, single picture forward or stills, an MV\_Next call must be issued.

This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which created the descriptor.

When playing from disc, the PCL parameter (a1) points to that PCL structure from the circularly linked chain of PCL's (Transfer Buffer), in which the first sector from the disc is to be expected. In practice, this corresponds to the address in the video CIL for the selected CD-I channel. If a buffer of which the PCL\_Sig field contains a signal value is emptied, this signal is sent to the application.

If a play on the given path is already queued, then this call will be queued by the kernel. If the process is currently playing through the same path, the active play will be aborted and the new play will be started. If the process has an active play through some other path, or if the given path is active because another process started a play on that path, then a E\$DevBsy will be returned. If some other process has a play active on a different path, then this MV\_Play request will be queued. The next play in the queue will be activated as soon as the active play is released through a MA\_Release call.

The first MV\_Play call after the MPEG Video decoder was released (or after a cold start) will allocate the MPEG Video buffer. If that memory is in use (modules are resident), a E\$NoRAM error is returned.

If, for a non-still map, the speed parameter is not equal to zero (scan, slow motion or single step), the application must take care of providing the correct data at the correct moment (i.e. doing a play from the CD and pause/continue that dataflow).

If loopback variables are set (relative to the beginning of the map) , these will be tested before it starts the play. If these variables are not correct (not within the offered map), an E\$IIIPrm error is returned. Otherwise playback starts at the starting loopback point. Loopback variables will have no effect when playing in scan mode.

---

IX.8 CD-RTOS for Full Motion

---

Each MV\_Play call will enable the video. If MV\_Show was issued before, a previously defined window will become visible.

If the type of the descriptor is still-image, the data is decoded into the still page as indicated by the MD\_Type field of the descriptor. The speed parameter is ignored. The decoder handles this play as if it were playing a sequence in single step speed. The first picture is decoded into the specified page (an illegal page number will result in an E\$IIPrm error). An MV\_Next call requests the decoder to reconstruct the following picture (if any) and to store into the page specified by that MV\_Next call.

Playing non-still type maps destroys the contents of the still pages in the decoder.

A play is finished:

- 1) When playing from CD, an MV\_Abort **must** be issued. When running out of data the application must decide whether it is the end of the play or that it is temporarily out of data.
- 2) When playing from Host, an MV\_Abort **may** be used. When the decoder runs out of data the play will also be finished when the last available picture is displayed (LPD signal).
- 3) Still pictures are treated as a normal sequence in single step mode.

If the sync mode parameter is set to -1, then the decoder assumes that no synchronization to MPEG Audio is required. This mode is called the **Non-synchronized mode**. If the sync mode parameter is set to -2, then this play will enter a wait state. It will remain in this state until an audio play, that must synchronize to this video path, has been started. This mode is called the **Wait mode**.

In **synchronized** mode, this parameter contains the path number of the active MPEG Audio play to which this MPEG Video play should synchronize itself. In the synchronized mode, no queuing will be done. Any play request while the decoder is in the synchronized mode will result in E\$DevBsy. The play that is active on the (sync) path to which this function wants to synchronize, should have been started by the same process and user. If not, a permission error will result.

The **synchronization offset** parameter indicates the constant difference between the timing parameters in the audio and video sequence. This parameter is defined, in units of 22.5 kHz, as the most significant 32 bits of the difference between the decoder system clocks in the MPEG Video decoder and the MPEG Audio decoder. In a formula:

$dsc(video) - dsc(audio)$ . See also Chapter IX.4.6.2.2.

IX.8 CD-RTOS for Full Motion

---

The ASYBlock pointer points to an asynchronous statusblock as defined in Chapter IX.8.2.3.3. Also the format of the speed parameter is described in Chapter IX.8.2.3.3; the usage of the other parameters is described in the MV\_ChSpeed definition.

Input:	d0.w = path number
	d1.w = MV_Play setstat function code
	d2.w = Video mapID
	d3.l = Offset to start
	d4.l = Host: Size of map (bytes)
	CD: Must be zero
	d5.l = Speed value
	d6.w = Sync mode/audio path to synchronize to
	d7.l = Synchronization offset to audio
	(a1) = Host: Start address of video map
	CD: Address of PCL structure
	(a4) = ASYblk pointer or null
Output:	None
Error output:	cc = carry bit set
	d1.w = error code
Possible errors:	E\$BPNum, E\$UnID, E\$DevBsy, E\$IIIPrm, E\$MemFul, E\$NoRAM, E\$BPAddr, E\$Permit

### 8.2.3.1.19 MV\_Pos - Set position of window

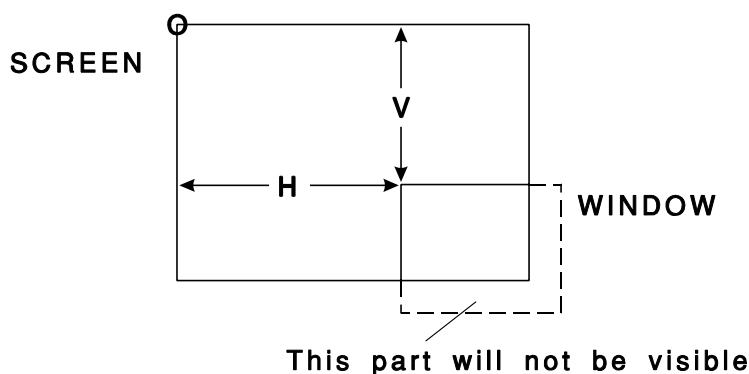
This function sets the position of the window on the full screen image, relative to the origin as set with MV\_Org.

If the specified map is currently active, the new parameters will become effective on the next picture change. If the scroll parameter (d5.b) is not zero and the window move is over less than 16 pixels and less than 16 lines, the window will be repositioned on the next vertical retrace.

This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which created the descriptor.

Input:

- d0.w = path number
- d1.w = MV\_Pos setstat function code
- d2.w = MapID
- d3.l = Offset (H:V) in UCM coordinates relative to screen origin.
- d5.b = scroll flag (not 0: scroll, 0: no scroll)



Output: None

Error output:

- cc = carry bit set
- d1.w = error code

Possible errors: E\$BPNuM, E\$UnID, E\$IIIPrM, E\$Permit

IX.8 CD-RTOS for Full Motion

---

**8.2.3.1.20 MV\_Release - Release the MPEG Video decoder for other processes**

When an MPEG Video play has finished, the process should release the decoder for other processes. As long as the decoder is not released, other processes that issued an MV\_Play command will remain in the sleeping queue.

If a map is active when the MV\_Release function is called, the play will be aborted. Implicitly, the synchronized mode, if enabled, will end because of this call. The MPEG Audio playback, that is part of this synchronized play, will continue in non-synchronized mode, without MPEG video.

If the display output was not yet turned off, the MV\_Release call will disable the output of the display. If the decoder was already released, a bad mode error will be returned.

This is a privileged call and may be used only by a process with a userID of the super user or, if the given path has an active play, by a process with the same userID as that of the process which started the play.

This call will also return the MPEG Video buffer to the operating system when no more plays are queued, otherwise the play that is queued first will be started.

Input:           d0.w = path number  
                  d1.w = MV\_Release setstat function code

Output:           None

Error output:    cc    = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$BMode, E\$Permit, E\$BPAddr

**8.2.3.1.21 MV\_SelStrm - Select an MPEG Video stream**

When another stream must be selected, this call passes the required parameters to the decoder. The sequence parameters must be passed together with this call to prevent a mismatch of those parameters with the selected stream.

This function will become effective on a picture change. At the same moment any data available in the input buffer of the MPEG Video decoder is skipped. The incoming data not belonging to the selected stream is skipped also.

The constraints for the image size parameter can be found in the MV\_ImgSize function description.

If the parameter Picture rate is zero, the decoder will skip all data until a sequence header was found.

This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which created the descriptor.

Input:           d0.w = path number  
                   d1.w = MV\_SelStrm setstat function code  
                   d2.w = MapID  
                   d3.b = MPEG Video stream number (0..15)  
                   d4.l = Image size (W:H) UCM coord.  
                   d5.b = Picture rate (0, 23..25, 29..30)

Output:           None

Error output:    cc    = carry bit set  
                   d1.w = error code

Possible errors: E\$BPNum, E\$UnID, E\$IIIPrm, E\$Permit

**8.2.3.1.22 MV\_Show - Enable the display of the window**

This function enables the window to be displayed in the full motion video plane. If an MPEG Video play is currently active then the window of the active map will be enabled on the next picture change, otherwise it is enabled on the next vertical retrace.

The page number parameter indicates the page to be displayed when not active or when the active play is of type still.

This is a privileged function. Permission is granted if the function is called by the super user, if no play is currently active or if the currently active play was started by the same user that calls this function.

Input:           d0.w = path number  
                  d1.w = MV\_Show setstat function code  
                  d2.b = page number

Output:           None

Error output:    cc    = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNuM, E\$Permit, E\$IIPrm



**8.2.3.1.23 MV\_SLink - Link subroutine module**

This function is used as a generalized method of expanding the command set of MoviMan. It utilizes a special subroutine module which is linked into MoviMan. When MoviMan receives an unknown GetStat or SetStat service request, it will pass the call to the subroutine module. If it is unknown to the subroutine module, the subroutine module should then pass the call to the driver.

There are four significant entry points into this external subroutine module: Init, SetStat, GetStat and Close.

Init will be called when the module is first linked to by MoviMan. It is provided to allow the module to initialize data structures and variables. Likewise, the Close entry point will be called when the last path to MoviMan is closed. At this time the subroutine module should free any memory which was allocated during its execution.

The SetStat and GetStat entry points are provided to actually do the main work of the subroutine module.

The mechanism for this dispatching should be similar to the one currently used by CD-RTOS file managers and drivers. The execution entry point of the module points to the primary jump table. This jump table contains the relative addresses for the Init, SetStat, GetStat and Close routines, in that order. These offsets are 2 byte entries.

When MoviMan wishes to pass control to one of these functions it will retrieve the appropriate address from the jump table and jump to that code.

Space is reserved in the MoviMan device static storage for this subroutine module to use. Eighty bytes are available, beginning at the offset V\_EXMOD.

It is impossible for two different subroutine modules to be linked at the same time. This applies even when they are linked by different processes. However, more than one process may be linked to the same subroutine module.

For more information, see A-VII.1

IX.8 CD-RTOS for Full Motion

---

Input:           d0.w = path number  
                  d1.w = MV\_SLink setstat function code  
                  (a0) = pointer to the name of subroutine module

Output:           None

Error output:    cc    = carry bit set  
                  d1.w = error code

Possible errors: E\$KwnMod, E\$BPNum, E\$MNF, E\$BNam, E\$ModBsy

**8.2.3.1.24 MV\_Status - Return the status of an active MPEG Video play**

This function returns the currently active MapID and its status. It passes a buffer which is filled in by the decoder. If no map is active, an E\$NoPlay error will result. If (a0) is a null-pointer, no status block will be filled and only the currently active MapID is returned.

By using the returned MapID value, more information can be retrieved by issuing the MV\_Info call and reading the descriptor fields.

This is a privileged call and may be used only by a process with a userID of the super user or, if a play is active, with the userID of the process which started that play.

Input:           d0.w = path number  
                  d1.w = MV\_Status getstat function code  
                  (a0) = pointer to the MV-status block or null

Output:           d0.w = MapID of active MVM

Error output:    cc    = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$NoPlay, E\$Permit

The format of the motion video status block is specified in Chapter IX.8.2.3.3.

**8.2.3.1.25 MV\_Trigger - Define MPEG Video events to signal**

This function activates signalling of MPEG Video events. The application can define, by means of setting or clearing bits in the event mask parameter, on the occurrence of which MPEG Video events it wants to receive a signal from the decoder. In case where one or more of the indicated events happens, a signal will be received. The value of this signal consists of two parts. The upper 5 bits of the 16-bit signal value are determined by the application at the moment it issued the MV\_Trigger call. The remaining bits reflect the status of the decoder at the time the event occurred. The decoder will only pass those status bits to the application that were enabled in the eventmask at the time the MV\_Trigger call was made.

The setting of the signal/eventmask remains valid for this path until either the path is closed or a new MV\_Trigger call is issued for this path.

Input:           d0.w = path number  
                  d1.w = MV\_Trigger setstat function code  
                  d3.w = signal/eventmask

Output:           None

Error output:    cc    = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum

Figure IX.8.4 **Signal/eventmask format**

	signal base	◆	NIS	BUF	EOS	EOI	CNP	LPD	SOS	GOP	PIC	DER		
Bit	15	..	11	10	9	8	7	6	5	4	3	2	1	0

- Bit 0: DER = signal when Data ERror detected  
 1: PIC = signal on PICture displayed  
 2: GOP = signal when display of the first intra-coded picture of a Group Of Pictures starts  
 3: SOS = signal when display of the first intra-coded picture of a sequence starts  
 4: LPD = signal when Last Picture Displayed  
 5: CNP = signal when old PCL-structure not in use anymore  
 6: EOI = signal when ISO 11172 End Code detected  
 7: EOS = signal when Sequence End Code detected  
 8: BUF = signal when Buffer UnderFlow detected  
 9: NIS = signal when new sequence parameters are found  
 10: Reserved and must be zero  
 15..11: Signal base: upper 5 bits of 16-bit signal to send (value must be between 00001 and 11111 binary)

The **PIC**, **GOP**, **SOS**, **LPD** and **NIS** events are synchronized to the video display process. They are generated in the vertical blanking interval in between active video periods (see Chapter IX.4.6.3).

An **EOI** or an **EOS** event is generated when an `iso_11172_end_code` or a `sequence_end_code` respectively is detected at the input of the MPEG Video Decoder.

The **NIS** event indicates that one or more of the values stored in the `MD_ImgSz` or `MD_PicRt` fields of the MPEG Video Descriptor have been changed by the Full Motion system because new values have been found in the MPEG Video stream by the MPEG Video Decoder. Current values (if any) are still available in the `MV_Status` block fields `MVS_ImgSz` and `MVS_PicRt`. See Chapter IX.8.2.3.3.1 and Chapter IX.8.2.3.3.3.

---

**IX.8 CD-RTOS for Full Motion**

---

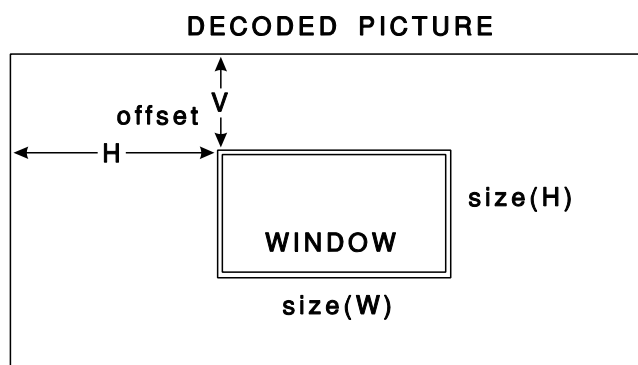
During normal speed play, slow motion and single step mode, the **NIS** event is generated at the display picture change prior to the change to the picture with the new parameter values. At the beginning of a play and in scan mode, the **NIS** event will be generated at least two video display fields before the change to the picture with the new parameter values.

The **LPD** event indicates that the last picture of an MPEG Video sequence is about to appear on the output of the MPEG Video Decoder. Generally this is the last picture - in display order - before a `sequence_end_code` of an MPEG Video stream. There is one exception though. When playing from host, the **LPD** event is also generated just before the last picture of the map or the looparea is displayed.

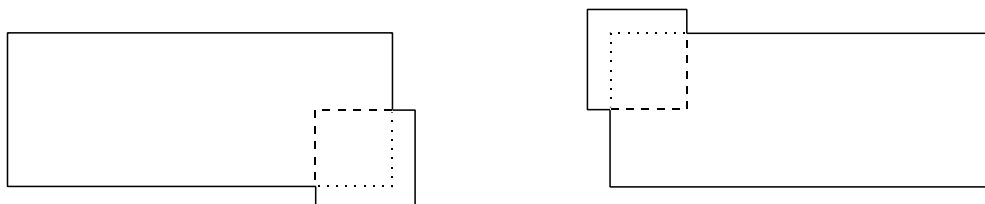
### 8.2.3.1.26 MV\_Window - Define display window within decoded picture

This function defines the display window within the decoded picture. It does not enable the display of the window. If the size and offset make the window come partly outside the decoded picture, the size will be adapted. If the window is positioned completely outside the decoded picture, an illegal parameter error will be returned. This means automatically that the decoded picture size must be known before the MV\_Window call.

Figure IX.8.5 Display window



Two exceptional cases, where the window comes partly outside the decoded picture, are illustrated below. In both cases, the decoder will adapt the size of the window.



In this case the sum of the offset and the size is greater than the size of the decoded picture. Only the area enclosed by dotted lines will be displayed.

In this case, the offset is negative. Only the area of the window enclosed by dotted lines will be displayed.

If the specified map is currently active, the new parameters will become effective on the next picture change. If the scroll parameter (d5.b) is not zero and the window move is less than 16 pixels and less than 16 lines, the window will be repositioned on the next vertical retrace.

IX.8 CD-RTOS for Full Motion

---

If the MVM is not active, the parameters are copied into the MVM descriptor and will be executed as soon as the MVM becomes active.

When the size of the decoded picture changes due to new sequence parameters, the window size will be adapted on the next picture change if the window is larger than the decoded picture. If the window is smaller than the decoded picture, no adaptation is done.

This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which created the descriptor.

Applications must be aware that the return value (the actual size) is only valid if a play is active and actual sequence parameters are set.

Input:	d0.w = path number
	d1.w = MV_Window setstat function code
	d2.w = MapID
	d3.l = Offset (H:V) in UCM coordinates
	d4.l = Size (W:H) in UCM coordinates
	d5.b = scroll flag (not 0: scroll, 0: no scroll)
Output:	d0.l = actual window size (W:H) in UCM coordinates
Error output:	cc = carry bit set
	d1.w = error code
Possible errors:	E\$BPNuM, E\$IIIPrm, E\$UnID, E\$Permit



**8.2.3.1.27 SS\_Opt - Read or write the options section**

Depending on the entry I\$GetStt or I\$SetStt, this function reads, respectively writes the option section of the path descriptor. The options section of the path descriptor contains information which may be of use to the application.

Input:           d0.w = path number  
                  d1.w = SS\_Opt get-/setstat function code  
                  (a0) = pointer to 128-byte option block

Output:           None

Error output:    cc    = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum

### 8.2.3.2 Path Handling Functions

Before any call can be issued to the MPEG Video decoder, a path to the MPEG Video decoding device must be opened. The calls supported by the CD-RTOS manager for path handling are `I$Open`, `I$Create` and `I$Close`.

#### 8.2.3.2.1 `I$Open` and `I$Create` - Open a path

`I$Open` and `I$Create` are synonymous functions since the MPEG video decoder is not a multi-file device. Therefore we will treat both calls as one and the same.

This call will set up all the necessary data structures to be able to access the motion video decoder. The path number that will be returned when the open call is successful must be used in all calls on this path.

Since the device is not a multi-file device, the path name consists of the device name preceded by a '/'.  
Example: `/MPEG`

Input:            `d0.b` = access mode (zero)  
                  `(a0)` = pointer to pathname

Output:           `d0.w` = path number  
                  `(a0)` = updated past pathname

Error output:    `cc`    = carry bit set  
                  `d1.w` = error code

Possible errors: `E$PthFul`, `E$BPNam`, `E$FNA`, `E$PNNF`

**8.2.3.2.2        I\$Close - Close a path**

This function closes the path identified by the path number parameter. If the MPEG Video decoder is allocated to this path the device will be released. If the calling process is the last using this path, all descriptors related to this path are released. If one of them is currently active, the play will be aborted provided that I\$Close is either called by the super user or by the user that started the play or if this is the last use of the path.

If a play is aborted that was in synchronized mode, the synchronized mode will end and the audio playback will continue in non-synchronized mode.

If the path closed was the last path to the MPEG Video decoder, the device is detached.

Input:                d0.w = path number

Output:              None

Error output:        cc     = carry bit set  
                      d1.w = error code

Possible errors: E\$BPNum

## IX.8 CD-RTOS for Full Motion

## 8.2.3.3 Data Structures

Several data structures are used to exchange information between the application and the MPEG Video decoder. Each of these structures is described in the following. All fields of these structures contain binary values and are positive unless specified otherwise.

## 8.2.3.3.1 MPEG Video Descriptor

Figure IX.8.6 MPEG Video Descriptor

Offset	Length	Name	Description
0	2	MD_Id	ID of the MVM
2	2	MD_Type	Type of this map
4	2	MD_Stream	Currently selected stream
6	4	MD_StLoop	Start address of loop
10	4	MD_EnLoop	End address of loop
14	2	MD_LpCnt	Initial loopback count
16	2	MD_LCntr	Current loopback count
18	4	MD_ImgSz	Size of the image (W:H)
22	4	MD_DecWin	Window in the image (W:H)
26	4	MD_DecOff	Offset window to decode (H:V)
30	4	MD_ScrOrg	Origin offset window (H:V)
34	4	MD_ScrOff	Offset within screen (H:V)
38	4	MD_BCol	Border color
42	4	MD_Speed	Display speed
46	4	MD_TimeCd	Picture time code (H:M:S:P)
50	2	MD_TmpRef	Temporal reference
52	1	MD_PicRt	Current picture rate
53	11		Reserved

**MD\_Id** This field identifies the MPEG Video Map. It is used in many calls to indicate the map on which the operation should be done. It is also referred to as MapID.

**MD\_Type** Identifies the type of MVM. For the specification of this field see the MV\_Create function (Chapter IX.8.2.3.1.7).

**MD\_Stream** Contains the number of the selected MPEG Video stream.

IX.8 CD-RTOS for Full Motion

---

- MD\_StLoop** **MD\_EnLoop** When playing from host, loopback points may be used. These points are addresses within the MVM, relative to the start of the MVM. These parameters are set using the call `MV_Loop` (see Chapter IX.8.3.1.13).
- MD\_LpCnt** This is the initial loop count, set by the `MV_Loop` call. The area between `MD_StLoop` and `MD_EnLoop` is repeated `MD_LpCnt` times. This field is not updated when doing the loopback.
- MD\_LCntr** This field indicates the number of loops remaining. Initially, it is set equal to `MD_LpCnt` (by the `MV_Loop` call). When reaching the end of the loop area this field is decremented. If the value becomes zero, the loop function is ended and playback will be aborted. At the end of the loop function the following fields will be set to zero: `MD_StLoop`, `MD_EnLoop`, `MD_LpCnt` and `MD_LCntr`.
- MD\_ImgSz** This field contains the size of the decoded picture. This size is fixed for a sequence. When a sequence is played, this field is maintained by the MPEG Video decoder. At the occurrence of the NIS event, this field is updated.  
The format of this field is according the conventions used in the UCM functions.
- MD\_DecWin** The display window is the rectangular area out of the decoded picture which can be displayed. The horizontal and vertical size of this window are defined in this field. These sizes must be smaller than or equal to the picture size as found in the `MD_ImgSz` field. This field can be set by issuing the call `MV_Window`. UCM conventions are assumed.
- MD\_DecOff** The position of the upper left corner of the window to extract from the decoded picture is set by this field. The offset relative to the upper left corner of the decoded picture is expressed in UCM coordinates. Both the horizontal and vertical offsets should be smaller than the picture size. The offsets may be negative, provided that the sum of these offset and the sizes is positive.
- MD\_ScrOrg** The origin specified in this field is an offset from the default origin which is the upper left corner of the full motion video plane. It is set by the `MV_Org` call where UCM conventions are assumed.

IX.8 CD-RTOS for Full Motion

---

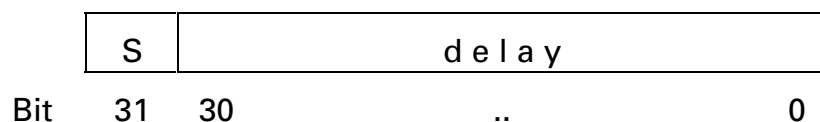
- MD\_ScrOff** The position of the display window in the full motion video plane is defined by this field. The position is considered as an offset relative to MD\_ScrOrg. The sum of the origin and this offset should be smaller than the size of the full motion video plane. The offset may be negative, provided the sum of this offset, the origin and the window size is positive. This field is set by the MV\_Pos function and is also formatted according to the UCM conventions.
- MD\_BCol** The format of the border color value is specified in Chapter 'MV\_BColor'. This function should be used to set this field.
- MD\_Speed** The format of the speed value is specified separately in this section. This field is only changed by the MV\_Play function or by the MV\_ChSpeed function.
- MD\_TimeCd** This field contains the time code of the first displayed picture of the current Group Of Pictures. The format is H:M:S:P where H stands for hours, M for minutes, S for seconds and P for picture which will vary from 0 to the picture rate minus one. In this field the bits 31..24 contain the value H, bits 23..16 contain the value M, bits 15..8 contain the value S and bits 7..0 represent the value P.
- MD\_TmpRef** The temporal reference is also taken from the MPEG Video stream when a picture is displayed.
- MD\_PicRt** The picture rate is another sequence parameter taken from the MPEG Video stream when found. It may have the values 23, 24, 25, 29 or 30, representing the picture rates of 23.976, 24, 25, 29.97 and 30 respectively. At the occurrence of a NIS event this field is updated.

IX.8 CD-RTOS for Full Motion

---

The following table presents the initial values of the descriptor fields (initiated by MV\_Create call) and a list of calls that may change these fields. When a play is aborted by MV\_Abort, MV\_Release, MV\_Show or another MV\_Play call, some fields are reset to their initial values.

Name	Initial value	Changeable by	Changed if aborted
MD_Id	free index #	fixed	no
MD_Type	User setting	fixed	no
MD_Stream	0	MV_SelStrm	no
MD_StLoop	0	MV_Loop	yes, set to 0
MD_EnLoop	0	MV_Loop	yes, set to 0
MD_LpCnt	0	MV_Loop	yes, set to 0
MD_LCntr	0	MV_Loop	yes, set to 0
MD_ImgSz	0:0	decoder/MV_ImgSize/MV_SelStrm	no
MD_DecWin	0:0	decoder/MV_Window	no
MD_DecOff	0:0	MV_Window	no
MD_ScrOrg	0:0	MV_Org	no
MD_ScrOff	0:0	MV_Pos	no
MD_BCol	0	MV_BColor	no
MD_Speed	0	MV_Play/MV_ChSpeed	no
MD_TimeCd	0	decoder	no
MD_TmpRef	0	decoder	no
MD_PicRt	0	decoder/MV_ImgSize/MV_SelStrm	no

**8.2.3.3.2 Speed Parameter**Figure IX.8.7 **Speed Parameter**

Four types of speed are distinguished:

1. If the speed value is zero (all 32 bits are 0), normal speed is assumed.
2. If bit S is set to one, the speed indicates scan mode. Only Access Pictures will be displayed. The time between two of these entrypoints is defined in multiples of 10 milliseconds. The minimal value depends on the seek time of the player, while the maximum value is application dependent. If, for example, bits 0 to 30 represent the value 250, a delay of 2.5 seconds ( $250 * 10 \text{ ms}$ ) will be between two consecutive entrypoints.
3. If bit S is zero and the delay value (bits 30..0) are in the range 2..8, slow motion is assumed. All pictures will be displayed at the specified slow motion speed. The delay value is not expressed in milliseconds but as a factor relative to the normal speed. This factor should be in the range [2..8]. The resulting display speed varies from 1/2th to 1/8th of the normal speed.
4. If bit S is zero and the delay value (bits 30..0) are all ones, the single step mode is assumed. In this mode, the next picture of a sequence is displayed at the request of the application.



## IX.8 CD-RTOS for Full Motion

8.2.3.3.3 **MV\_Status block**

The format of the MV\_Status block, as returned by MV\_Status:

Figure IX.8.8 **MV\_Status block**

Offset	Length	Name	Description
0	2	MVS_LCntr	Loops remaining
2	4	MVS_CurAdr	Address to retrieve data from
6	4	MVS_Speed	Display speed
10	4	MVS_ImgSz	Size of the image (W:H)
14	4	MVS_TimeCd	Picture timecode
18	2	MVS_TmpRef	Temporal reference of picture
20	2	MVS_Stream	Active stream number
22	1	MVS_PicRt	Current picture rate
23	1		Reserved
24	4	MVS_DSC	Video decoder system clock
28	4		Reserved

Apart from the **MVS\_DSC** field, the values of all **MVS\_xxx** fields are defined in their **MD\_xxx** equivalents in Chapter IX.8.2.3.3.1. All the bits in all fields but **MVS\_LCntr** and **MVS\_CurAdr** will be set to 1 as long as the decoder is not started. The same fields will not be updated when the play is frozen, paused or directly after a new stream is selected and no data is decoded yet. Additionally, the following definitions are used:

**MVS\_CurAdr** When playing from host, this field contains the address within the MVM (relative to the start of the MVM) where the MPEG Video system will take the data from to copy to the Fifo. When playing from disc, the value will be zero.

**MVS\_Speed** The format is defined in the speed parameter format description in this section.

**MVS\_Stream** All bits of this field will be 1 after a new stream is selected and no data is decoded yet.

**MVS\_DSC** This field contains the 31 most significant bits of the current value of the decoder system clock. This 31 bit value is unsigned. The most significant bit in this field is always zero.

## IX.8 CD-RTOS for Full Motion

8.2.3.3.4 **Asynchronous Status Block**Figure IX.8.9 **Asynchronous Status Block**

Offset	Length	Name	Description
0	2	ASY_Stat	Current status of the operation
2	2	ASY_Sig	Signal to be sent on termination

**ASY\_Sig** This field contains the signal number to be sent to the application when the operation finishes or a fatal error occurs. It is initialized by the application before the operation is started and may be changed by the application during the operation. If this field is set to zero then no signal is sent when the operation finishes or an error occurs. If the play is aborted by an error situation or by the application (MV\_Abort), this field will contain the error code. The play is finished when the given signal is sent and the play finished bit is set in ASY\_Stat.

**ASY\_Stat** During an MPEG video play, these bits are copied from the hardware status. It is the applications responsibility to clear the status bits in the ASY\_Stat field before the play is started.

The layout of the ASY\_Stat field is given in figure IX.8.10.

Figure IX.8.10 **Layout of the ASY\_Stat field**

Bit #	Description	Event
0	Operation is finished	
1	PICTure	PIC
2	Group Of Pictures	GOP
3	Sequence header found	SOS
4	Last Picture Displayed	LPD
5	Changed to new PCL-structure	CNP
6	End Of ISO stream	EOI
7	End Of Sequence	EOS
8	Buffer underflow detected	BUF
9	New sequence parameters detected	NIS
10..14	Reserved (must be zero)	
15	Error has occurred	DER

IX.8 CD-RTOS for Full Motion

---

The **PIC**, **GOP**, **SOS**, **LPD** and **NIS** bits are set to one when the corresponding events are generated (see Chapter IX.8.2.3.1.25) and reset to zero at the next picture change.

The **CNP** bit is set to one when the system does not use the old PCL anymore and it will be set to zero when a new PCL pointer is accepted by the MPEG video system.

Bits **EOI** and **EOS** are set to one when the corresponding events are generated (see Chapter IX.8.2.3.1.25). They are reset to zero when data from the next MPEG pack is sent to the MPEG Video decoder.

Bit **BUF** is set to one when the decoder runs out of data (when decoding) and is set to zero on the next data transfer.

The following combinations of the bits 0 and 15 mean:

Bit 0	Bit 15	Meaning
0	0	Not started or playing
0	1	Error detected, continued playing
1	0	Play finished normally
1	1	Fatal error occurred, error code in ASY_Sig

### 8.2.3.3.5 Error Structure

This format is identical to the structure in Chapter VII. When playing from CD, the usage of this structure is specified in the description of the SS\_Play function in Chapter VII.2.2.3.2. If this structure is used to conceal errors in a Host map with the MV\_Conceal call, it should be initialized identically.

Figure IX.8.11 Error Structure format

Offset	Length	Name	Description
0	4	Err_BufSiz	Total size of error buffer
4	1	Err_Res	Resolution (b/w) of error info
5	1		Reserved
6	2	Err_Cnt	No of blocks containing errors
8	2	Err_Offset	Offset to first error data block (e.g. 26+N)
10	8		Reserved for decoder use
18	8		Reserved for application use
26	N	Err_Blocks	Block error map
26+N	M		First error data block

## 8.2.4 MPEG Audio Functions

The MPEG Audio functions available to the application are described in this section. With each function, its purpose, its constraints and its register usage (for input and for output) are described. Any data structures, used for communication between application and moviman, are described also.

### 8.2.4.1 The Setstat and Getstat Functions

The setstat and getstat functions are, in alphabetical order, described in this subsection.

**8.2.4.1.1 MA\_Abort - Abort the current MPEG audio play**

This function aborts the play that is currently being executed. The play will not be active any longer. This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which started the play. If no play is active, an abort error will be returned.

A successful call causes the output to be muted (attenuator) and the fields in the MPEG Audio descriptor are reset (see Chapter IX.8.2.4.3).

If this play was running in **synchronized** mode with a video play, then this call will end the synchronized mode. The video playback will continue in non-synchronized mode.

Input:           d0.w = path number  
                  d1.w = MA\_Abort setstat function code

Output:           None

Error output:   (cc) = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$Abort, E\$Permit

**8.2.4.1.2 MA\_Close - Free the audio descriptor**

This function returns the given descriptor to the decoder. This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which created the audio mapID.

If the given descriptor was playing (or paused) at the time of this call, the play will be aborted before freeing the descriptor.

If another process has a play-request, using this descriptor, queued, then that process will be activated to return an E\$UnID error.

Input:           d0.w = path number  
                  d1.w = MA\_Close setstat function code  
                  d2.w = Audio mapID

Output:           None

Error output:   (cc) = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$UnID, E\$Permit

IX.8 CD-RTOS for Full Motion

---

8.2.4.1.3 **MA\_Cntrl - Select MPEG audio stream and set attenuators**

This function selects for the given mapID the MPEG audio stream to be decoded. The attenuation at the time of playback is also set with this call. The attenuation parameter is the same as with the SC\_Atten function (see Chapter VII). The parameters will be stored in the descriptor. If the ID is active at the time of the call, then the parameters are used immediately. This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which created the audio mapID.

If, on an active play, it is switched to another MPEG audio stream in the ISO 11172 stream, it can take some time before this new stream is actually decoded. The moment that this new stream becomes active can be monitored in two ways:

- The MA\_Status call shows which stream is being decoded
- The MA\_Trigger call can be used to signal the event where the new stream becomes active.

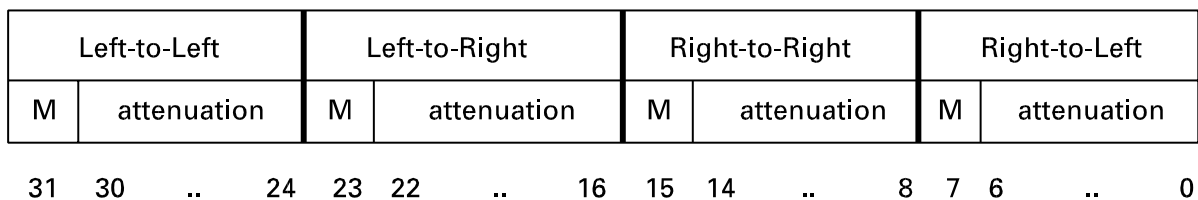
Input:               d0.w = path number  
                       d1.w = MA\_Cntrl setstat function code  
                       d2.w = Audio mapID  
                       d3.l = Attenuation value  
                       d4.w = Audio stream to decode (0..31)

Output:             None

Error output:      (cc) = carry bit set  
                       d1.w = error code

Possible errors: E\$BPNum, E\$UnID, E\$Permit

Figure IX.8.12 **Attenuation value format**



where M = '1': mute (no sound)  
           '0': use attenuation value.

**8.2.4.1.4 MA\_Continue - Continue a paused MPEG Audio play**

This function continues a paused play. This is a privileged call and may be used only by a process with a userID of the super user or with the userID of the process which issued the pause command. If the play is not paused, or if no play is active, a bad mode error will be returned.

The availability of audio data is the responsibility of the application. When playing from CD, data retrieval must be started after this call. When playing from host, data retrieval is resumed by the MPEG audio decoder. When playing from host, decoding will resume immediately. When playing from CD, decoding will resume as soon as the MPEG decoder system timing allows to start decoding of the data coming from disc.

If the audio play is active in **synchronized** mode with video, and if this synchronized play is paused, this function will cause both the audio and the video to continue. If a synchronous play is frozen, this function will return an error (E\$BMode).

Input:           d0.w = path number  
                  d1.w = MA\_Continue setstat function code

Output:           None

Error output:   (cc) = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$BMode, E\$Permit



## IX.8 CD-RTOS for Full Motion

8.2.4.1.5 **MA\_Create - Create an MPEG Audio Descriptor**

This call reserves and initializes the MPEG Audio descriptor. The format of the descriptor, the initial values and usage of its fields can be found in Chapter IX.8.2.4.3.

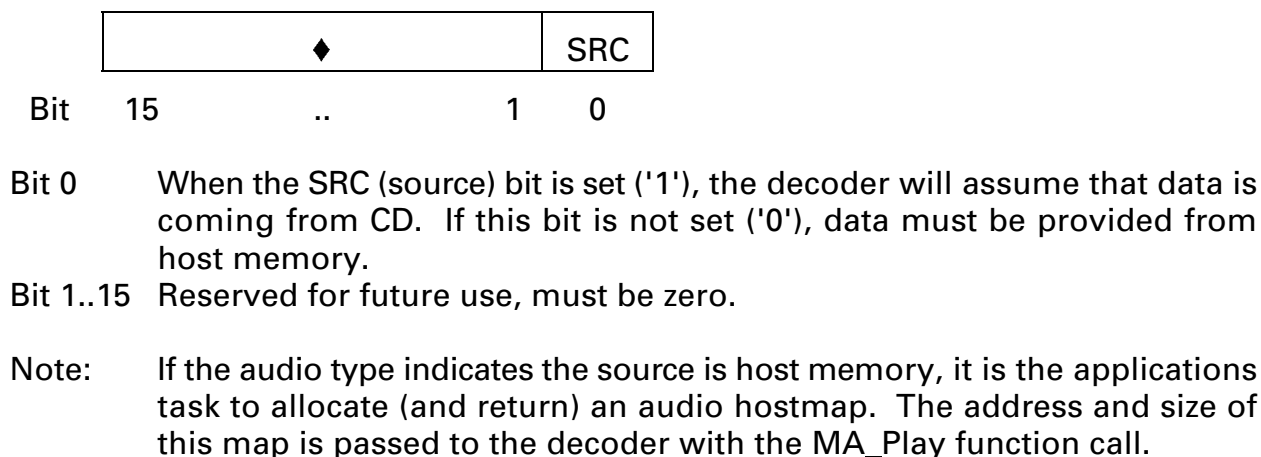
Input:           d0.w = path number  
                   d1.w = MA\_Create getstat function code  
                   d2.w = Audio type

Output:           d0.w = Audio mapID

Error output:   (cc) = carry bit set  
                   d1.w = error code

Possible errors: E\$BPNum, E\$IIIPrm, E\$NoRam, E\$MemFull

Figure IX.8.13 **Format of the audio type parameter**



**8.2.4.1.6 MA\_Info - Return the pointer to the MPEG audio descriptor**

MA\_Info returns a pointer to the MPEG audio descriptor corresponding to the given audio mapID. The fields in the descriptor are for information purposes only. Fields should only be changed by calling the appropriate functions.

This is a privileged call and may be used only by a process with the userID of the super user or with the userID of the process which created this MPEG audio descriptor.

Input:           d0.w = path number  
                  d1.w = MA\_Info getstat function code  
                  d2.w = Audio mapID

Output:           (a0) = pointer to audio descriptor

Error output:   (cc) = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$UnID, E\$Permit

**8.2.4.1.7 MA\_Jump - Compensate audio streams' timing parameters on behalf of seamless jump**

The application is allowed, when playing from CD, to discontinue the current data stream and to start supplying data that comes from another position. To overcome a discontinuity in timing parameters, this function provides the application a means to adapt the new timing parameters in such a way that they fit seamlessly into the old stream. As soon as either a play is started or the decoder detects a discontinuity in the supplied stream, it will start correcting the timing parameters with a given delta value. This delta value is defined as  $SCR(j) - SCR(i)$  and is given in a resolution of 22.5 kHz.  $SCR(j)$  is the value of the first sector of the new data and  $SCR(i)$  is the value of the SCR of the sector that would have been supplied next if no jump was done. Initially the delta value is zero. After a play is aborted, the delta value will be reset to zero. When playing from host, the delta value will have no influence at all. Since a discontinuity in SCR is detected only if the difference in between SCR's is negative or greater than 0.70 seconds, a positive delta value less than or equal to 15,750 is considered as an illegal parameter.

Input:           d0.w = path number  
                   d1.w = MA\_Jump setstat function code  
                   d3.l = Delta value (signed)

Output:           None

Error output:   (cc) = carry bit set  
                   d1.w = error code

Possible errors: E\$BPNum, E\$IIIPrm

#### 8.2.4.1.8 MA\_Loop - Repeat a part of the MPEG audio map

This function sets the loopback points. This is a privileged call and may be used only by a process with the userID of the super user or with the userID of the process which created this audio descriptor.

This call is only allowed for audio mapID's of the 'host' type. For other types, a bad mode error will be returned. If the audio map is not active, setting the loopback variables will have no immediate effect, apart from copying the loop parameters into the descriptor. As soon as the map is activated, the loopback function will be executed. The map IN MEMORY has to start with a pack header. The Relative loopback startpoint must contain an MPEG Audio Pointer (See Chapter IX.5.4.3.2). The Relative loopback endpoint must point to the byte following the last audio frame to be included in the loopback area.

The start and end are offsets relative to the start of the map.

If the audio map (with it's loopback points set) is activated (through MA\_Play), it will play the audio data from the start of the looparea until the end of the looparea is reached. The Loopcount will be decremented and when it did not reach zero, the audio data between the looppoints will be decoded again until the Loopcount is zero. When the Loopcount reaches zero, the playback will be aborted.

If the audio map is already being played when this call is made, the behavior of this function depends on the current position in the audio map. If the current position is before the ending loopback point, then playing continues until the ending loopback point is reached, at which time the Loopcount is decremented and the looping area is played again, if necessary. If the current position is already past the ending loopback point, an immediate jump will take place to the starting loopback point.

If the MA\_Loop call is applied to a play which is already in loopback mode, then the currently active loop area will be completed first (i.e. played until the end position of the active loop area). At that moment, the loop parameters of the most recent MA\_Loop call will become active. How the play continues depends on the current position (which is the end position of the 'old' loop area). See the previous paragraph.

If the designated mapID is active and in **synchronized** mode with video, then this MA\_Loop call will result in a E\$BMode error.

IX.8 CD-RTOS for Full Motion

---

Input:           d0.w = path number  
                  d1.w = MA\_Loop setstat function code  
                  d2.w = Audio mapID  
                  d3.l = Relative loopback start point (bytes)  
                  d4.l = Relative loopback end point (bytes)  
                  d5.w = Loopcount

Output:           None

Error output:   (cc) = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNuM, E\$UnID, E\$BMode, E\$IIIPrm, E\$Permit

**8.2.4.1.9 MA\_Pause - Pause the current MPEG Audio play**

This function pauses the MPEG Audio play that is currently being executed. This is a privileged function call and may be used only by a process with a userID of the super user or with the userID of the process which started the play operation. If no MPEG Audio play is active at the time this function is called, or if the play is already paused, a bad mode error will be returned.

If the audio play is in **synchronized** mode with video, this function will pause both the audio and the video playback. If a synchronous play is frozen, this function will return an error (E\$BMode).

Input:           d0.w = path number  
                  d1.w = MA\_Pause setstat function code

Output:           None

Error output:   (cc) = carry bit is set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$BMode, E\$Permit

#### 8.2.4.1.10 MA\_Play - Start an MPEG audio play

This function starts to play the data belonging to the given MPEG audio mapID. The data may be coming from disc or from host memory. Play is an asynchronous operation, i.e. the application continues execution at the same time as the play is executed.

This is a privileged call and may be used only by a process with the userID of the super user or with the userID of the process which created the MPEG audio descriptor.

The Offset to start parameter must contain an MPEG Audio Pointer (See Chapter IX.5.4.3.2).

When playing from disc, the PCL parameter (a1) points to that PCL structure from the circularly linked chain of PCL's (Transfer Buffer), in which the first sector from the disc is to be expected. In practice, this corresponds to the address in the audio CIL for the selected CD-I channel. If a buffer of which the PCL\_Sig field contains a signal value is emptied, this signal is sent to the application.

If a play on the given path is already queued, then this call will be queued by the kernel. If the process is currently playing through the same path, the active play will be aborted and the new play will be started. If the process has an active play through some other path, or if the given path is active because another process started a play on that path, then a E\$DevBsy will be returned. If some other process has a play active on a different path, then this MA\_Play request will be queued. The next play in the queue will be activated as soon as the active play is released through a MA\_Release call.

If loopback variables are set, these will be tested before the play is started. If these variables are not correct i.e. not within the offered map, an E\$IllPrm error is returned. Otherwise playback starts at the starting loopback point.

If the sync mode parameter is set to -1, then the decoder assumes that no synchronization to video is required. This mode is called the **Non-synchronized mode**. If the sync mode parameter is set to -2, then this play will enter a wait state. It will remain in this state until a video play has been started, that must synchronize to this audio path. This mode is called the **Wait mode**.

IX.8 CD-RTOS for Full Motion

---

In **synchronized** mode, this parameter contains the path number of the active video play to which this audio play should synchronize itself. In the synchronized mode, no queuing will be done. Any play request while the decoder is in the synchronized mode will result in E\$DevBsy. The play that is active on the (sync) path to which this function wants to synchronize, should have been started by the same process and user. If not, a permission error will result.

If the video play is paused or in a non-normal speed, then the audio will be started anyway, but no audible output will be generated. As soon as the accompanying video starts to run in normal speed, the audio will be enabled. The **synchronization offset** parameter indicates the constant difference between the timing parameters in the audio and video sequence. This parameter is defined, in units of 22.5 kHz, as the most significant 32 bits of the difference between the decoder system clocks in the MPEG Video decoder and the MPEG Audio decoder. In a formula:

$$\text{dsc}(\text{video}) - \text{dsc}(\text{audio}) \text{ (See Chapter IX.4.6.2.2)}$$

Input:	d0.w = path number d1.w = MA_Play setstat function code d2.w = Audio mapID d3.l = Offset to start d4.l = Host: Mapsize (bytes) CD: Must be zero d6.w = Sync mode/video path to synchronize to d7.l = Synchronization offset to video (a1) = Host: Start address of audio map CD: Address of PCL structure (a4) = Asyblock pointer for audio events
Output:	None
Error output:	(cc) = carry bit set d1.w = error code
Possible errors:	E\$BPNum, E\$UnID, E\$BMode, E\$DevBsy, E\$IIIPrm, E\$Permit



**8.2.4.1.11 MA\_Release - Release the MPEG Audio decoder for other processes**

When an MPEG Audio play has finished, the process should release the MPEG audio decoder for other processes. As long as the decoder is not released, other processes that issued an MA\_Play command will remain in the sleeping queue.

If an MPEG audio play is still active on the given path when the MA\_Release function is called, the play will be aborted. Implicitly, the **synchronized** mode, if enabled, will end because of this call. The MPEG video playback, that is part of this synchronized play, will continue in non-synchronized mode, without MPEG audio.

If the userID of the caller is not the super user and is not equal to the userID of the process that started that play, then aborting will fail and result in a permission error.

If the audio output was not yet muted, the MA\_Release call will mute the output of the audio. If the device is not allocated to the path, then a bad mode error will be returned.

Input:           d0.w = path number  
                  d1.w = MA\_Release setstat function code

Output:           None

Error output:   (cc) = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$BMode, E\$Permit

**8.2.4.1.12 MA\_SLink - Link subroutine module**

This function is used as a generalized method of expanding the command set of MoviMan. It utilizes a special subroutine module which is linked into MoviMan. When MoviMan receives an unknown GetStat or SetStat service request, it will pass the call to the subroutine module. If it is unknown to the subroutine module, the subroutine module should then pass the call to the driver.

There are four significant entry points into this external subroutine module: Init, SetStat, GetStat and Close.

Init will be called when the module is first linked to by MoviMan. It is provided to allow the module to initialize data structures and variables. Likewise, the Close entry point will be called when the last path to MoviMan is closed. At this time the subroutine module should free any memory which was allocated during its execution.

The SetStat and GetStat entry points are provided to actually do the main work of the subroutine module.

The mechanism for this dispatching should be similar to the one currently used by CD-RTOS file managers and drivers. The execution entry point of the module points to the primary jump table. This jump table contains the relative addresses for the Init, SetStat, GetStat and Close routines, in that order. These offsets are 2 byte entries.

When MoviMan wishes to pass control to one of these functions it will retrieve the appropriate address from the jump table and jump through to that code.

Space is reserved in the MoviMan device static storage for this subroutine module to use. Eighty bytes are available, beginning at the offset V\_EXMOD.

It is impossible for two different subroutine modules to be linked at the same time. This applies even when they are linked by different processes. However, more than one process may be linked to the same subroutine module.

For more information, see A-VII.1

IX.8 CD-RTOS for Full Motion

---

Input:           d0.w = path number  
                  d1.w = MA\_SLink setstat function code  
                  (a0) = pointer to the name of subroutine module

Output:           None

Error output:    cc    = carry bit set  
                  d1.w = error code

Possible errors: E\$KwnMod, E\$BPNum, E\$MNF, E\$BNam, E\$ModBsy

**8.2.4.1.13 MA\_Status - Return the status of the current MPEG Audio play**

This function returns the currently active audio mapID and its status. If no map is active, an E\$NoPlay error will result. If the active play was not started with a userID equal to that of the caller and the caller is not the super user, then a permission error will be returned.

A 32-byte buffer has to be passed which is filled in by the decoder. If (a0) is a null-pointer, no status block will be filled in and just the currently active MapID is returned.

By using the returned audio mapID value, more information can be retrieved by issuing the MA\_Info call and reading the descriptor fields.

Input:           d0.w = path number  
                  d1.w = MA\_Status getstat function code  
                  (a0) = pointer to the MA\_Status block or null

Output:           d0.w = Active audio mapID

Error output:   (cc) = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum, E\$NoPlay, E\$Permit

The format of the MA\_Status block is given in Chapter IX.8.2.4.3.

**8.2.4.1.14 MA\_Trigger - Define MPEG audio events to signal**

This function activates signalling of MPEG audio events. The application can define, by means of setting or clearing bits in the event mask parameter, on the occurrence of which events it wants to receive a signal from the MPEG Audio decoder.

In case where one or more of the indicated events occur, a signal will be received. The value of this signal consists of two parts. The upper 5 bits of the 16-bit signal value are determined by the application at the moment it issued the MA\_Trigger call. The remaining bits reflect the status of the decoder at the time the event occurred. The system will only pass those status bits to the application that were enabled in the eventmask at the time the MA\_Trigger call was made.

The setting of the signal/eventmask remains valid for this path until either the path is closed or a new MA\_Trigger call is issued for this path.

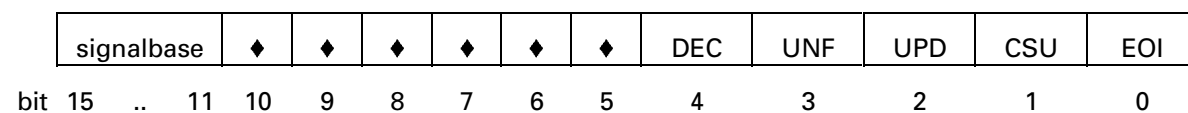
Input:           d0.w = path number  
                  d1.w = MA\_Trigger setstat function code  
                  d3.w = signal/eventmask

Output:           None

Error output:   (cc) = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum

## IX.8 CD-RTOS for Full Motion

Figure IX.8.14 **Format of signal/eventmask parameter**

where

- bit 0        EOI    ISO 11172 End Code detected
- bit 1        CSU    Decoder changed to a new audio stream
- bit 2        UPD    Decoder updated the frame header (see MA\_Status)
- bit 3        UNF    Decoder does not have data to decode (underflow)
- bit 4        DEC    Decoder started decoding
- bit 5..10    Reserved, should be zero
- bit 15..11   Signal base: upper 5 bits of 16-bit signal to send (value must be between 00001 and 11111 binary)

**8.2.4.1.15 SS\_Opt - read or write option section**

Depending on the entry I\$GetStt or I\$SetStt, this function reads, respectively writes the option section of the path descriptor. The option section of the path descriptor contains information which may be of use to the application.

Input:           d0.w = path number  
                  d1.w = SS\_Opt get-/setstat function code  
                  (a0) = pointer to 128 byte buffer

Output:           None

Error output:   (cc) = carry bit set  
                  d1.w = error code

Possible errors: E\$BPNum

### 8.2.4.2 Path Handling Functions

Before any call can be issued to the MPEG Audio decoder, a path must be opened to the decoder. The calls supported by the CD-RTOS manager for path handling are `I$Open`, `I$Create` and `I$Close`.

#### 8.2.4.2.1 `I$Open` and `I$Create` - open a path

The calls `I$Open` and `I$Create` are synonymous, since the MPEG Audio decoder is not a multi-file device. Therefore both calls are treated as one and the same. This call will set up all the necessary data structures to enable access to the MPEG Audio decoder. The path number that will be returned when this call is successful must be used in all subsequent calls on this path.

Since the device is not a multi-file device, the path name consists of the device name preceded by a '/'.

Input:                `d0.b` = access mode (zero)  
                      `(a0)` = pointer to pathname

Output:              `d0.w` = path number  
                      `(a0)` = updated past pathname

Error output:        `(cc)` = carry bit set  
                      `d1.w` = error code

Possible errors: `E$PthFul`, `E$BPNam`, `E$FNA`, `E$PNNF`



**8.2.4.2.2        I\$Close - close a path**

This call closes the path identified by the path number parameter. If the MPEG Audio decoder is allocated to this path, then the device will be released. An active play on this path will be aborted if one of the following conditions is true:

- I\$Close is called by the super user,
- I\$Close is called by the same user that started the active play,
- This is the last use of the path (i.e. does not exist as a duplicated or inherited path anymore).

Input:                d0.w = path number

Output:              None

Error output:        (cc) = carry bit set  
                      d1.w = error code

Possible errors: E\$BPNum

### 8.2.4.3 Data Structures

Several data structures are used to exchange information between the application and the MPEG audio decoder. Each of these structures is described in the following. All fields in these structures contain binary values and are positive unless specified otherwise.

#### 8.2.4.3.1 MPEG Audio Descriptor

In the following two tables the fields of the MPEG audio descriptor are specified as well as the initial value of each field and the value of a field after an abort.

Figure IX.8.15 MPEG Audio Descriptor

Offset	Length	Name	Description
0	2	MD_Id	ID of the MPEG audio map
2	2	MD_Type	Type of this map
4	2	MD_Stream	Selected MPEG stream (0..31)
6	4	MD_StLoop	Loopback start point
10	4	MD_Enloop	Loopback end point
14	2	MD_LpCnt	Initial value loopback count
16	2	MD_LCntr	Actual value loopback count
18	1	MD_At_LL	Attenuator left-to-left
19	1	MD_At_LR	Attenuator left-to-right
20	1	MD_At_RR	Attenuator right-to-right
21	1	MD_At_RL	Attenuator right-to-left
22	10		Reserved

**MD\_Id** This field identifies the MPEG audio descriptor. It is used in many calls to indicate the map on which the operation should be done. It is also referred to as MPEG audio mapID.

**MD\_Type** Identifies the type of MPEG audio map. This field is the same as specified in the MA\_Create call.

IX.8 CD-RTOS for Full Motion

---

MD_Stream	The selected MPEG audio stream (0..31) can be found in this field. When changing stream via the MA_Cntrl function, this field is updated immediately, although it may take some time before the desired stream is decoded. The MAS_Stream field in the MA_Status block indicates the stream number that actually is being decoded. A signal, corresponding to the CSU event (see MA_Trigger), indicates the actual switch to a new stream.
MD_StLoop MD_EnLoop	When playing from host, loopback points may be used. These points are offsets within the MPEG audio map, relative to the start of that map. These parameters are set using the MA_Loop call (see Chapter IX.8.2.4.1.8).
MD_LpCnt	This is the initial loop count, set by the MA_Loop call. The area between MD_StLoop and MD_EnLoop is repeated MD_LpCnt times. This field is not updated when doing the loopback.
MD_LCntr	This field indicates the number of loops remaining. Initially, it is set equal to MD_LpCnt (by the MA_Loop call). When reaching the end of the loop area this field is decremented. If the value becomes zero, the loop function is ended and playback will be aborted. At the end of the loop function the following fields will be set to zero: MD_StLoop, MD_EnLoop, MD_LpCnt and MD_LCntr.
MD_At_LL	This field contains the attenuation parameter for the left-to-left audio path. The layout of this field is as shown in the MA_Cntrl call. The value in this field is applied as soon as this map becomes active. If the decoder decides to mute the output of this map (e.g. as part of the MA_Abort function), the actual status of the attenuator can be found in the MA_Status block (see MA_Status function). During MA_Create this field is initialized to hex 0x80, no attenuation and muted.
MD_At_LR	Same as for MD_At_LL, but for the left-to-right path. Initial value is 0xFF, maximum attenuation and muted.
MD_At_RR	Same as for MD_At_LL, but for the right-to-right path. Initial value is 0x80, no attenuation and muted.
MD_At_RL	Same as for MD_At_LL, but for the right-to-left path. Initial value is 0xFF, maximum attenuation and muted.

## IX.8 CD-RTOS for Full Motion

The following table presents the initial values of the MPEG Audio descriptor fields (initialized by MA\_Create call) and a list of calls that may change these fields. When a play is aborted, some fields are reset to their initial values.

Name	Initial value	Changeable by	Changed if aborted
MD_Id	free index #	fixed	no
MD_Type	User setting	fixed	no
MD_Stream	0	MA_Cntrl	no
MD_StLoop	0	MA_Loop	yes, set to 0
MD_EnLoop	0	MA_Loop	yes, set to 0
MD_LpCnt	0	MA_Loop	yes, set to 0
MD_LCntr	0	MA_Loop	yes, set to 0
MD_At_LL	0x80, muted	MA_Cntrl	yes, set to 0x80
MD_At_LR	0xFF, muted	MA_Cntrl	yes, set to 0xFF
MD_At_RR	0x80, muted	MA_Cntrl	yes, set to 0x80
MD_At_RL	0xFF, muted	MA_Cntrl	yes, set to 0xFF

**8.2.4.3.2 MA\_Status Block**

The format of the MA\_Status block is given below.

Figure IX.8.16 **MA\_Status Block**

Offset	Length	Name	Description
0	2	MAS_Stream	Actually decoded MPEG Audio stream
2	4	MAS_Att	Actual attenuator status
6	4	MAS_Head	Audio frame header information
10	4	MAS_CurAdr	Current offset in hostmap
14	4	MAS_DSC	Audio decoder system clock
18	14		Reserved

**MAS\_Stream** This field contains the number of the MPEG Audio stream which is currently being decoded by the MPEG Audio decoder. This stream number may differ from the number passed with the last MA\_Cntrl function call as long as the requested stream is not found. As soon as audio packets with the requested stream are available for the decoder, this field is updated to contain the requested stream number. As long as the decoder is not decoding any stream, because the selected stream is not yet available, all bits in this field will be 1.

**MAS\_Att** This field contains the actual value of the attenuator. This value does not necessarily have to be the same as set by the user. On some occasions (e.g. MA\_Abort) the decoder may decide to change the attenuator settings to prevent annoying clicks. The settings as used by the decoder can be found in this field. The layout of this field is as described in the MA\_Cntrl function.

**MAS\_Head** Each audio frame contains a header with information on the nature of the audio stream. The layout and meaning of the fields in this packed 32-bit statusword are explained in the following table. Before the first header arrives at the decoder, all bits in this field will be 1.

**MAS\_CurAdr** This field contains during playback of MPEG audio from host, the current position, relative to the start of the map. When playing from CD, this field remains zero.

IX.8 CD-RTOS for Full Motion

---

**MAS\_DSC** This field contains the 31 most significant bits of the current value of the decoder system clock. This 31 bit value is unsigned. The most significant bit in this field is always zero. Before the decoder receives MPEG Audio data, this field will be -1. If the play is paused, the field will not be updated until new data is copied to the decoder due to the MA\_Continue function.

## IX.8 CD-RTOS for Full Motion

In the following table the layout and meaning of the fields in the packed 32-bit status-word of MAS\_Head are explained.

Figure IX.8.17 MAS\_Head status word

Name	#bits	From..To	Description
RES	12	31..20	Reserved
ID	1	19	'1' MPEG coded
LAY	2	18..17	coded algorithm layer '10' layer II '11' layer I
PROT	1	16	'0' redundancy for error checking '1' no redundancy added
RATE	4	15..12	Bit-rate index
SF	2	11..10	Sample frequency '00' 44.1 kHz
PAD	1	9	Padding flag '0' No additional slot added '1' Additional slot to adjust frame
PRIV	1	8	Private bit
MODE	2	7..6	Mode, see also next field '00' Stereo '01' Joint stereo (see MODEXT) '10' Dual channel '11' Single channel
MODEXT	2	5..4	Mode extension for 'joint stereo' '00' subbands 4-31 intensity stereo '01' subbands 8-31 intensity stereo '10' subbands 12-31 intensity stereo '11' subbands 16-31 intensity stereo
CR	1	3	Copyright indication '0' Not copyright protected '1' Copyright protected
ORG	1	2	Home copy '0' or original '1'
EMPH	2	1..0	Type of de-emphasis to use '00' no emphasis '01' 50/15 $\mu$ s emphasis

Note that for the CD-I Full Motion system not all combinations from ISO 11172 are allowed, see also Chapter IX.5.3.2 (MPEG Audio Parameters).

8.2.4.3.3 **Asynchronous Status Block (ASY-block)**Figure IX.8.18 **Asynchronous Status Block**

Offset	Length	Name	Description
0	2	ASY_Stat	Current status of the operation
2	2	ASY_Sig	Signal to be sent on termination

**ASY\_Stat** During an MPEG Audio play these bits are copied from the MPEG Audio decoder status. It is the applications responsibility to clear the ASY\_Stat field before the play is started. The **UNF** bit is set to one if, during decoding, the decoders buffer runs out of data. The bit will be set to zero at the next data transfer. The **DEC** bit is set to one as long as the decoder is decoding. The **OVF** bit is set to one if, during a data transfer, the decoders buffer has no room for this data. If such an overflow occurs, the play will be aborted.

Figure IX.8.19 **The layout of the ASY\_Stat field**

Bit	Description
0	Operation finished
1..2	◆
3	Buffer underflow (UNF)
4	Decoding (DEC)
5	Buffer overflow (OVF)
6..14	◆
15	Fatal Error



IX.8 CD-RTOS for Full Motion

---

**ASY\_Sig** This field contains the signal number to be sent to the application when the audio operation finishes. It is initialized by the application before the operation is started and may be changed by the application during the operation. If this field is set to zero then no signal is sent when the operation finishes. If the play finishes because of an error (bit 15 in ASY\_Stat field is set), then the ASY\_Sig field will be filled with an appropriate errorcode:

- in case of overflow: E\$Write
- in case of an abort: E\$Abort
- in case of DMA error: E\$Read, E\$NotRdy or E\$BusErr

### 8.2.5 MPEG Still Picture Functions

To control MPEG Still Picture decoding, the same functions and data structures are available as for MPEG Video decoding.

### 8.3 Device Drivers

#### 8.3.1 Basic Functional Requirements of Moviman Drivers

The Moviman device driver module is a package of five subroutines that are called by Moviman or by the kernel in system state. The functions are:

- (1) Initialize the device controller hardware and related driver variables as required.
- (2) Return a specified device status.
- (3) Set a specified device status.
- (4) De-initialize the device. It is assumed that the device will not be used again unless re-initialized.
- (5) Handle the interrupts, generated by the device controller hardware.

When written properly, a single physical driver module can handle multiple identical hardware interfaces. The specific information for each physical interface (port address, initialization constants, etc.) is given in a device descriptor module. Device descriptor modules are described in Chapter IX.8.3.2.1.

The name by which the device is known to the system is the name of the device descriptor module. CD-RTOS copies the information contained in the device descriptor module to the path descriptor data structure and the device driver static storage area for easy access by the drivers.

### 8.3.2 Data Structures

MoviMan and its drivers must concern themselves with the following data structures:

- (1) Device Descriptors
- (2) Path Descriptors
- (3) Device Static Storage
- (4) Device Drive Tables
- (5) Process Descriptors
- (6) Map Descriptor Structures
- (7) Play Control Lists

In the following paragraphs, structures (1) through (4) are described in more detail. The other structures are described:

- |                        |                                       |
|------------------------|---------------------------------------|
| (5) Process Descriptor | See A-VII.1                           |
| (6) Map Descriptor     | See 3.1., IX-8.2.3.3. and IX-8.2.4.3. |
| (7) Play Control List  | See VII-2.2.3.2.                      |

#### 8.3.2.1 MoviMan Device Descriptors

Device descriptor modules are small, non-executable modules that provide information that associates a specific I/O device with its logical name, hardware controller address(es), device driver name, file manager name, and initialization parameters.

Device drivers and file managers both operate on general classes of devices, not specific I/O ports. The device descriptor modules tailor their function to a specific I/O device. One device descriptor module must exist for each I/O device in the system. However, one device may also have several device descriptors with different initialization constants.

The name of the module is used as the logical device name by the system and the user. Its format consists of a standard module header (48 bytes) that has a module type code "Devic". The remaining header fields (24 bytes) are standard for device descriptor modules.

IX.8 CD-RTOS for Full Motion

---

**8.3.2.1.1 Standard Device Descriptor Header Fields**

The first 72 bytes of each device descriptor are defined in the same way for all device descriptors in CD-RTOS.

**8.3.2.1.2 Device Descriptor Option Fields (Initialization Table)**

The initialization table is copied into the 'option section' of the path descriptor when a path to the device is opened. It is generally referenced using the path descriptor. Additional option fields may be added here. The values in this table may be used to define the operating parameters that are accessible by the I\$GetStt and I\$SetStt system calls of SS\_Opt. The maximum size of the initialization table is 128 bytes.

Figure IX.8.20 **Device Descriptor Option Fields**

Offset	Length	Name	Description
72	1	PD_DTP	Device Class (= 13)
73	1	PD_DRV	Device Number
74	1	PD_MDL	Map descriptor length (in bytes) for audio 32, for video 64
75	1	PD_TYP	Device Type (0=Video, 1=Audio)
76	4	PD_MCL	Color of decoder memory (or zero)
80	4	PD_MSZ	Size of decoder memory (or zero)
84	4	PD_ADR	Address of decoder memory (or zero)

**8.3.2.1.3 Device Configuration Fields**

There is a section of the device descriptor reserved for Device Configuration variables (the DevCon section). There are no fields defined for the DevCon section of MoviMan device descriptor.

### 8.3.2.2 Moviman Path Descriptors

Every open path is represented by a data structure called a path descriptor. Every time an `I$Open` system call is invoked, a path descriptor is created. It contains information required by file managers and device drivers to perform I/O functions. Path descriptors are dynamically allocated and de-allocated as paths are opened and closed.

Path descriptors have three sections, a standard section, a file manager section, and an options section.

#### 8.3.2.2.1 Standard Path Descriptor Fields

The first 42 bytes of the Path Descriptor are defined in the same way for all path descriptors in CD-RTOS. These fields may be read by the drivers, but may not be modified by them.

Figure IX.8.21 Standard Path Descriptor Fields

Offset	Length	Name	Description
0	2	PD_PD	Path number
2	1	PD_MOD	Mode (read/write/update)
3	1	PD_CNT	Number of paths using this P.D.
4	4	PD_DEV	Device Table entry address
8	2	PD_CPR	Current process ID
10	4	PD_RGS	Caller's register stack pointer
14	4	PD_BUF	Buffer address
18	4	PD_USER	Group/User ID of path's creator
22	4	PD_Paths	Linked list of open paths on device
26	2	PD_COUNT	Actual number of paths using this Path descriptor
28	2	PD_LProc	Last active process ID
30	12		Reserved

### 8.3.2.2.2 File Manager Path Descriptor Fields

MoviMan defines the following fields for its own use. Drivers will read some of these fields but normally do not write them.

Figure IX.8.22 MoviMan Path Descriptor Fields

Offset	Length	Name	Description
42	2	PD_Signal	Signal base and eventmask (Trigger)
44	4	PD_SyncPath	User pathnumber to synchronize with
48	1	PD_Master	Used for synchronized looping, see below
49	1	PD_Ready	Used for synchronized looping, see below
50	8		Reserved
58	2	PD_MapID	MapID of play in queue
60	2		Reserved
62	4	PD_NxtPD	Next path descriptor pointer in queue
66	4	PD_PrVpd	Previous path descriptor in queue
70	4	PD_EXTMOD	Pointer to external subroutine module
74	2	PD_PROCID	Process ID of active process
76	4	PD_AsyBlk	Pointer to asynchronous status block
80	4	PD_OldPCL	Pointer to old PCL when switching
84	4	PD_NxtPCL	Pointer to PCL in broken 'old' PCL list
88	6		Reserved
94	4	PD_DTPtr	Drive table pointer
98	30		Reserved

The meaning of the fields PD\_Master and PD\_Ready is only defined when playing in synchronized loop mode.

PD\_Master = 1: This play is considered to act as 'master', i.e. the other play has to synchronize itself to this play.

PD\_Master = 0: This play is considered to behave as 'slave', i.e. this play has to synchronize itself to the other play.

PD\_Ready indicates that this play has reached the end of its loop area, and that it will wait until the other play has reached the end of its loop area. As soon as both play have reached the end of the loop area, both plays restart at the beginning of their loop areas and will reset the PD\_Ready field to zero.

**8.3.2.2.3 Option Table Path Descriptor**

The first fields of the option table are copied directly from the device descriptor initialization fields.

Figure IX.8.23 **Option Table of a MoviMan Path Descriptor**

Offset	Length	Name	Description
128	1	PD_DTP	Device Class (= 13)
129	1	PD_DRV	Device Number
130	1	PD_MDL	Map descriptor length (in bytes) for audio 32, for video 64
131	1	PD_TYP	Device Type (0=Video, 1=Audio)
132	4	PD_MCL	Color of decoder memory (or zero)
136	4	PD_MSZ	Size of decoder memory (or zero)
140	4	PD_ADR	Address of decoder memory (or zero)

**8.3.2.3 MoviMan Device Driver Static Storage**

MoviMan device driver modules contain a package of subroutines that take care of controlling MPEG decoders and of supplying MPEG data to those controllers. Because these modules are re-entrant, one copy of the module can simultaneously run several identical I/O controllers.

The kernel will allocate a static storage area for each device (which may control several decoders). The size of the storage area is given in the device driver module header (M\$Mem). Some of this storage area is required by the kernel and MoviMan. The device driver may use the remainder in any manner.

Figure IX.8.24 **MoviMan Device Driver Static Storage**

Offset	Length	Name	Description
0	4	V_PORT	Device base port address
4	2	V_LPRC	Last active process ID. This contains the process ID of the last process to use the device.
6	2	V_BUSY	Current Process ID (0 = not busy)
8	2	V_WAKE	Process ID to awaken. This contains the process ID of any process that is waiting for the device to complete I/O (0 = no process waiting). Maintained by the device driver.
10	4	V_PATHS	Linked list of open paths. This is a singly-linked list of all paths currently open on this device. It is maintained by the kernel.
14	32		Reserved
46	1	V_NDRV	Number of drives. This contains the number of drives that the controller can use. It is defined by the device driver as the maximum number of drives that the controller can work with. MoviMan will assume that there is a drive table for each drive.
47	1	V_WakeStat	Wake-up condition of unqueued process 0=still sleeping, 1=normal wakeup, 220=error
48	2	V_IRQMask	Mask to disable interrupts with
50	4	V_Descr	Pointer to first block of 16 map descriptors
54	4	V_ExtMod	Address of subroutine module (or zero)
58	4	V_ExtLnk	Link count for subroutine module
62	80	V_ExMod	Reserved data area for subroutine module
142			Drive tables. This contains one table per drive that the controller will handle. MoviMan will assume there are as many tables as specified in V_NDRV.

### 8.3.2.3.1 Drive Tables

After the driver's INIT routine has been called, MoviMan requests the driver to initialize the drive table for the corresponding drive. As mentioned in Figure IX.8.24 (last item), each drive has a corresponding drive table. The drive table format is as follows:



## IX.8 CD-RTOS for Full Motion

Figure IX.8.25 **MoviMan Drive Table Format**

Offset	Length	Name	Description
0	4	V_Path	Active path descriptor pointer
4	4	V_Proc	Active process descriptor pointer
8	4	V_CurDesc	Pointer to current map descriptor
12	2	V_PlayType	Type of Play (see below)
14	4	V_PCL	Current PCL pointer
18	4	V_Offset	Offset to start in pack
22	4	V_SkipSec	Sectors to skip until offset (CD)
26	4	V_EnLoop	Active end-loop offset
30	2	V_LCntr	Active loop counter
32	4	V_NewLpEnd	New Endaddress (offset) of loop area
36	4	V_LpStart	Startaddress (offset) of the loop area
40	4	V_MapPtr	Pointer to the start of the hostmap
44	4	V_MapSize	Size of the hostmap (bytes)
48	4	V_CurAdr	Offset to pack to be played next
52	2	V_SyncPath	Sync mode / path to synchronize with
54	4	V_SyncOffs	Synchronization offset (22.5 kHz)
58	1	V_Paused	Play paused flag (0=not paused)
59	1	V_Sync	Play is in sync mode (0=not in sync)
60	4	V_SCR	SCR emulation counter
64	4	V_ScanTime	Time to wait between displaying 2 frames
68	4	V_NewPCL	PCL to link to (automatically)
72	4	V_MemAddr	Address of allocated decoder memory
76	4	V_MemSize	Size of allocated decoder memory
80	1	V_Frozen	Picture frozen flag (0=not frozen)
81	1	V_VidOn	Video on flag (0=not on)
82	1	V_WinOn	Window on flag (0=not on)
83	1		Reserved
84	4	V_PCLwrite	Empty PCL pointer to write to
88	4	V_Window	Window size (W:H) in UCM coordinates
92	4	V_DecOff	Window offset (X:Y) in UCM coordinates
96	4	V_ScrOrg	Origin in screen (X:Y) in UCM coord.
100	4	V_ScrOff	Offset in screen (X:Y) in UCM coord.
104	4	V_BorCol	Border color (0:R:G:B)
108	4	V_Att	Audio attenuator value (LL:LR:RR:RL)
112	1	V_Waste	Audio in 'waste mode' flag (0=no waste)
113	1	V_Wait	Wait flag for drivers (0=no wait)
114	4	V_Speed	User selected speed value
118	4	V_CurDelta	Currently used delta for seamless jump
122	4	V_NewDelta	Delta to use after next jump in SCR
126	64		Reserved, workspace for drivers

IX.8 CD-RTOS for Full Motion

---

The definition of the V\_PlayType field:

- 0: Decoder is stopped
- 1: Decoder is playing (at normal speed)
- 2: Decoder is playing in scan mode
- 3: Decoder is playing in single step mode
- 4: Decoder is playing in slow motion mode

### 8.3.3 **MoviMan Device Driver Subroutines**

As with all CD-RTOS device drivers, MoviMan device drivers use a standard executable memory module format with a module type code "Drivr". The execution offset address in the module header points to a branch table that has seven entries. Each entry is the offset to a corresponding subroutine. The branch table is as follows:

Ent_tbl	dc.w INIT	Initializes device
	dc.w READ	(Not used)
	dc.w WRITE	(Not used)
	dc.w GETSTAT	Gets device status
	dc.w SETSTAT	Sets device status
	dc.w TERM	Terminates device
	dc.w TRAP	(Not used)

Each subroutine should exit with the condition code register carry bit cleared, i.e. set to zero, if no error occurred. Otherwise the carry bit should be set to one and an appropriate error code returned in d1.w.

IX.8 CD-RTOS for Full Motion

---

8.3.3.1 The INIT Subroutine

The INIT routine's function is to initialize the device and the associated static storage. The INIT routine must:

- (1) Initialize the device's permanent storage. This minimally consists of initializing V\_NDRV to the number of drives with which the controller will work. Also the V\_IRQMask should be initialized. Also a block of 16 mapdescriptors should be allocated and initialized. The address of this block has to be stored in V\_Descr. The layout of such a block is as follows:

Figure IX.8.26 Layout of a map descriptor block

Offset	Length	Name	Description
0	4	MD_Next	Pointer to next block of 16 (or NULL)
4	4	MD_Free	Pointer to 1st free descriptor in this block (or NULL)
8	2	MD_FreelD	ID of 1st free descriptor in this block
10	2		Reserved
12	4	MD_Alloc	Actual size of allocated block, required as parameter when returning this memory.
16	32	MD_Paths	An array of 16 words (2 bytes) corresponding to the 16 mapdescriptors in this block. Each entry in this array contains the system path number of the path through which the mapdescriptor was created (or -1)
48	64	MD_UserID	An array of 16 longwords (4 bytes) corresponding to the 16 mapdescriptors in this block. Each entry in this array contains the userID (Grp:Usr) of the process that created the mapdescriptor.
112			At this offset, 16 contiguous mapdescriptors are available. For the layout of such a descriptor, refer to IX.8.2.3.3 and IX.8.2.4.3.

All fields, also those in the 16 mapdescriptors, should have the initial value zero. The following fields have to be set to these initial values:

MD_FreelD	1
MD_Free	Startaddress + 112
MD_Alloc	Actual size (bytes)
MD_Path	All entries must be -1

IX.8 CD-RTOS for Full Motion

---

- (2) Initialize device control registers and enable interrupts.
- (3) Place the IRQ service routine on the IRQ polling list by using the F\$IRQ service request.

Note: Prior to being called, the device permanent storage will be cleared (set to zero) except for V\_PORT which will contain the device address. The driver should initialize each drive table.

Input:           (a1) = Device descriptor pointer  
                  (a2) = Device static storage pointer  
                  (a4) = Current process descriptor pointer  
                  (a6) = System global data storage pointer

Output:           None

Error Output: cc    = Carry bit set to one  
                  d1.w = Error code

### 8.3.3.2 The READ Subroutine

This subroutine is not supported in Moviman drivers. Any attempt to use the READ subroutine will return with the carry bit set to one and an error code of E\$UnkSvc in d1.w.

### 8.3.3.3 The WRITE Subroutine

This subroutine is not supported in Moviman drivers. Any attempt to use the WRITE subroutine will return with the carry bit set to one and an error code of E\$UnkSvc in d1.w.

### 8.3.3.4 The GETSTAT and SETSTAT Subroutines

These two subroutines are grouped together because of their similarity in use. These routines are used to get/set the device's operating parameters as specified for the I\$GetStt and I\$SetStt service requests.

#### 8.3.3.4.1 General requirements for GETSTAT and SETSTAT

When the driver's getstat or setstat entry point is called, the registers have the following values:

Input:	d0.w = Function code (a1) = Path descriptor pointer (a2) = Device static storage pointer (a4) = Current process descriptor pointer (a5) = Pointer to user's register stack (a6) = System global data storage pointer Other registers are function code dependent
Output:	Function code dependent
Error Output:	cc = Carry bit set to one d1.w = Error code

The specific parameters supplied by the application are generally passed to the drivers by means of the user register stack in register a5. Drivers can access these parameters by referring to R\$xx(a5), where xx is the name of the register.

#### 8.3.3.4.2 Additional requirements for GETSTAT Subroutines

The currently defined getstat functions (see IX.8) require no additional parameters.

#### 8.3.3.4.3 Additional requirements for SETSTAT Subroutines

The currently defined setstat functions (see IX.8) require no additional parameters. Apart from the functions mentioned in Chapter IX.8, the following setstat driver routines are available to the file manager:

**8.3.3.4.3.1 MA\_Waste - Stop and mute audio decoder, skip audio packs**

This function stops the audio decoder, mutes the attenuator and takes care that the drivers' ISR skips all data. This is valid for both play from disc (skip incoming sectors) as for play from host (disc emulation, skip packs). The play remains active. This function is called by the file manager when during a synchronized play, audio should be muted, e.g. when the corresponding video play changes to a non-normal speed. The waste mode is cleared by calling the MA\_Continue function.

Additional input: None

Output: None

Error output: (cc) = carry bit set to one  
d1.w = error code

Possible errors: E\$BMode

**8.3.3.4.3.2 MA\_ReqSync - Request to synchronize to active audio play**

This function can be considered as a hook in the driver for synchronization purposes. It requests the audio driver to supply timing information for a video play that wants to synchronize to an active audio play. How this information is exchanged is left to the implementation of the drivers. However, the timing information has to be made available as soon as possible, either from available MPEG audio data or from an internal drivers' variable.

The sync offset, required to set the correct timing value, is only passed as a parameter when a synchronized play has to be started, i.e. when the application calls the MV\_Play function with the sync parameters filled in. The driver has to store this offset internally. If a synchronized play is already running, the driver should use this internal variable. This case is indicated by the caller if with the MA\_ReqSync call d2.l is set to -1. The MA\_ReqSync function is called by the filemanager in the MV\_Play function (where d2.l contains the syncoffset parameter). It is also called in the Mx\_Continue function, if video has to synchronize to audio. In this latter case, d2.l should be -1.

Additional input: d2.l = syncoffset or -1

Output: None

Error output: (cc) = carry bit set to one  
d1.w = error code

Possible errors: E\$BMode



**8.3.3.4.3.3 MV\_ReqSync - Request to synchronize to active video play**

This function can be considered as a hook in the driver for synchronization purposes. It requests the video driver to supply timing information for an audio play that wants to synchronize to an active video play. How this information is exchanged is left to the implementation of the drivers. However, the timing information has to be made available as soon as possible, either from available MPEG video data or from an internal drivers' variable.

The sync offset, required to set the correct timing value, is only passed as a parameter when a synchronized play has to be started, i.e. when the application calls the MA\_Play function with the sync parameters filled in. The driver has to store this offset internally. If a synchronized play is already running, the driver should use this internal variable. This case is indicated by the caller if with the MV\_ReqSync call d2.l is set to -1. The MV\_ReqSync function is called by the filemanager in the MA\_Play function (where d2.l contains the syncoffset parameter). It is also called in the Mx\_Continue function, if audio has to synchronize to video. In this latter case, d2.l should be -1. Finally, the MV\_ChSpeed call where the new speed is 'normal', invokes this MV\_ReqSync function, with d2.l equal to -1.

Additional input: d2.l = syncoffset or -1

Output: None

Error output: (cc) = carry bit set to one  
d1.w = error code

Possible errors: E\$BMode

### 8.3.3.5 The TERMINATE Subroutine

This routine is called when a device is no longer in use in the system. This is defined as when the link count of its device table entry becomes zero.

The TERMINATE subroutine must:

- (1) Stop any ongoing actions on this device.
- (2) Disable the device interrupts.
- (3) Return the memory allocated for all mapdescriptors.
- (4) Remove the device from the IRQ polling list.

Input:           (a1) = Device descriptor pointer  
                  (a2) = Device static storage pointer  
                  (a4) = Current process descriptor pointer  
                  (a6) = System global data pointer

Output:           None

Error Output:   cc    = Carry bit set to one  
                  d1.w = Error code

**8.3.3.6 The TRAP subroutine**

The TRAP entry should be defined as the offset to the exception handling code or zero if no handler is available. This entry point is currently not used by the kernel.

**8.3.3.7 The IRQ Service Request Subroutine**

Although this routine is not included in the device driver module branch table and is not called directly by the MoviMan, it is a key routine in interrupt driven device drivers. Its general functions include:

- (1) Poll the device to determine the interrupt type. If the interrupt is not caused by the MPEG audio or video controller, the carry bit must be returned set to one with an RTS instruction as quickly as possible.
- (2) Service device interrupts.
- (3) When the IRQ service routine finishes servicing an interrupt it must clear the carry bit (i.e. set to zero) and exit with an RTS instruction.

In the interrupt service routine of the driver the following actions must be executed:

From the first data to copy to the decoder, the SCR value divided by 4 (22.5 kHz), must be set into the field V\_SCR in the static storage. This field must be updated regularly. For video, the update of the V\_SCR field must be according to the speed (e.g. 11.25 kHz for slow motion factor 2, or add picture rate (divided by 4) on every picture change when stepping). For audio, the increment of V\_SCR must be equal to the time interval since the previous update, also expressed in units of 22.5 kHz.

When playing from CD:

- (1) Poll the current PCL for data in a rhythm that is fast enough to read at least one sector every 1/75 of a second.
- (2) If data is found, do not copy that data unless the SCR in that pack is less than or equal to the V\_SCR value.
- (3) If the data is copied to the decoder, free the PCL and copy the pointer PCL\_Nxt in V\_PCL.  
If the PCL\_Sig of the copied PCL is set, send this signal to the process with ID = PD\_PROCID. See path descriptor as pointed to by V\_Path.
- (4) For video only:  
If the application requested a new PCL (e.g. when changing speed) link this new PCL chain into the PCL that was just copied (use PD\_OldPCL and PD\_NxtPCL to save the link pointers). If the current PCL is not in use anymore, restore the link and signal the application.

IX.8 CD-RTOS for Full Motion

---

When playing from host:

- (1) Do not copy the data with an SCR that is higher than the V\_SCR in the static storage.
- (2) When all data is copied to the decoder, do not end the play until all data was decoded.

For synchronization purposes:

The audio and video drivers must exchange each others V\_SCR values to be able to keep the play synchronized. When looping, wait until the other driver has finished its loop area, then restart the loop.

Input:                   (a2) = Device Static storage pointer  
                             (a3) = Port address  
                             (a6) = System global data storage pointer

Note: IRQ service routines may destroy the following registers only: d0, d1, a0, a2, a3 and a6. All other registers must be preserved.

# CD-I Full Functional Specification

## Appendix I

### Table of Contents

---

<b>A Glossary of Terms</b>	<b>Page</b>
A	1-3
B	4
C	5-9
D	10-12
E	13-14
F	15-16
G	17
H	18
I	19-20
K	21
L	22
M	23-24
N	25
O	26
P	27-29
Q	30
R	31-32
S	33-36
T	37
U	38
V	39
W	40
Y	41

Appendix I

---

This page is intentionally left blank

### Appendix I

#### A I Glossary of Terms

---

**Absolute Disc Address** - see Chapter III, page 1. The location of a given sector on the disc (in minutes, seconds and sectors) contained in the header of each sector in the main channel.

**Abstract File Name** - see Chapter III, page 10. Field of the File Structure Volume Descriptor identifying the path name of a file containing a summary of the volume contents; this file can contain audio and video data.

**Access Controller** - see Chapter VIII, page 3. See Memory Access Controller.

**Active Line Scan Period** - see Chapter V, page 9. The time taken by the electron beam of a cathode ray tube to move across the visible part of a line on the screen.

**Adaptive Delta Pulse Code Modulation** - see Chapter IV, page 14. A technique for converting analog audio into digital audio. Delta modulation assumes close correlation between successive samples. It cannot accurately express large transients in an audio signal, because the correlation between successive samples is too low. Adaptive delta pulse code modulation is a variant of delta modulation in which the quantization steps are adapted to the dynamic amplitude variation. This adaptation can include a temporary switch to PCM. See Pulse Code Modulation.

**Address of Path Table** - see Chapter III, page 9. The block address of the first block of the system path table.

**ADPCM** See Adaptive Delta Pulse Code Modulation.

**ADPCM Decoder** - see Chapter IV, page 21. Converts the CD-I audio sector encoded data to 2's Complement 16-bit PCM encoded audio.

**Album Identifier** - see Chapter III, page 9. Field of the File Structure Volume Descriptor identifying the set of discs (or album) to which the volume belongs.

**Album Set Sequence Number** - see Chapter III, page 9. Specifies the relative sequence number of a disc in the album to which it belongs.

**AMIN, ASEC, AFRAME** - see Chapter II, page 8. An absolute time scale addressing the whole of a Compact Disc and located in the subcode Q channel.

## Appendix I

### A I Glossary of Terms

---

**Anti-aliasing filter** - see Chapter V, page 19. When a signal to be sampled contains frequencies higher than half the sampling rate, aliasing will occur. That is, the reconstituted sampled signal will not contain these high frequency signals, but will contain false low frequency components instead. An anti-aliasing filter reduces aliasing by attenuating these high frequency components before sampling.

**Application Identifier** - see Chapter III, page 9. Field of the File Structure Volume Descriptor specifying the path name of the first application program to be executed when the disc is mounted.

**Application Specific Coding Flag** - see Chapter V, page 96. Indicates whether the video coding conforms to the specification of Chapter V of the CD-I Specification or is application specific, i.e. to be interpreted by the application software.

**ASCF** - see Chapter V, page 96. See Application Specific Coding Flag.

**Asynchronous Execution Service Request** - see Chapter VII, page 15. A service request where the process will continue to execute after the request has been made and before the request has been satisfied.

**ATIME** - see Chapter II, page 10. Absolute time on a disc. Expressed (as AMIN, ASEC, AFRAME) in 6-digit BCD in the Q-channel of the Compact Disc subcode.

**Attributes** - see Chapter III, page 22. Two bytes in a directory record that specify the permissions and characteristics of the corresponding file.

**Audio Block Field Bytes** - see Chapter IV, page 7. 2304 bytes of data in a CD-I audio sector. An audio block is further subdivided into 18 sound groups of 128 bytes each. The sound groups have to be encoded in time sequential order.

**Audio Channel** - see Chapter IV, page 3. A CD-I track can be divided into several audio channels, one for each audio source.

**Audio Channel Selection Register** - see Appendix II, page 3. Within an audio sector the channel number can be a value from 0 to 15. The audio channel selection register selects audio channels and causes their transfer to the audio processor.

**Audio Data Representation** - see Chapter IV, page 1. The way in which audio data is encoded and stored on a Compact Disc and decoded according to the CD-I specifications.



## Appendix I

### A I Glossary of Terms

---

**Audio Decoder Delay** - see Chapter VIII, page 9. The delay caused by the ADPCM decoder and audio processing unit, which should be less than 27 ms. This is the delay time between the audio data input to the ADPCM decoder and an audible output.

**Audio Functions** - see Chapter VII, page 58. Compact Disc File Manager functions which are concerned with the maintenance and manipulation of ADPCM sound.

**Audio Mixing Control Unit** - see Chapter IV, page 27. Consists of four attenuators between the two audio inputs and two audio outputs of the unit. The attenuation is controlled by the UCM in steps of 1 dB.

**Audio Processing Unit** - see Chapter IV, page 20. Converts digitally coded audio information into the left and right analogue outputs. Also includes an Audio Mixing Control Unit.

**Audio Sector** - see Chapter IV, page 4. A sector in which the data field contains audio data.

**Audio Sector Data Format** - see Chapter IV, page 4. The data field of an audio sector comprises a sub-header, an audio data block of 2304 bytes and 20 bytes with value 0.

**Audio Sector Interleaving** - see Chapter IV, page 13. A method of recording audio sectors with a fixed duty cycle on the disc, according to the average data rate required.

**Audio Track** - see Chapter II, page 3. A CD-DA track with digital audio samples encoded as 16-bit wide 2's complement numbers.

## Appendix I

### A I Glossary of Terms

---

**Backdrop** - see Chapter V, page 92. The background image plane that becomes visible when all or parts of all other planes are made transparent.

**Base Case System** - see Chapter VIII. The minimum characteristics always offered by any CD-I system.

**Bit Mapped Text** - See Chapter V, page 18. Text which is not program-generated in the decoder but which is stored on the disc in image form.

**Block Address** - see Chapter III, page 1. A 32-bit integer that is converted to an absolute disc address to access information on the disc.

**Breadth First Traversal** - see Chapter III, page 24. A method of parsing a tree structure in which each level is completely parsed before moving to each successive level.

### Appendix I

#### A I Glossary of Terms

---

**Cartesian Coordinate System** - see Chapter VII, page 83. A system for locating a point in a plane by specifying its distance from two axes which intersect at right angles.

**Cartoon-style image** - see Chapter V, page 18. An image containing significant areas of the same color which can be efficiently run length coded: this is generally true for cartoon images.

**CCIR** - see Appendix V, page 1. Comité Consultatif International Radio

**CD-DA Controller Decoder** - see Chapter VIII, page 3. The hardware needed to control the playing of a CD-DA or CD-I disc and to route the information coming from the disc, either in the form of CD-DA audio information or other information, for further decoding.

**CD-DA Data** - see Chapter II, page 37. Data encoded according to the CD-DA specification.

**CD-DA Track** - see Chapter I, page 2. A track on a Compact Disc containing audio information encoded according to the CD-Digital Audio specification.

**CDFM** See Compact Disc File Manager.

**CDFM Device Driver** - see Chapter VII, page 225. The lowest level of system software which provides the interface between the CDFM and the CD controller.

**CD-I Decoder Model** - see Chapter II, page 37. The general structure of the decoding unit for decoding CD-I disc information containing four independent units.

**CD-I Encoder Model** - see Chapter II, page 35. A generalized structure of the encoding process.

**CD-I Media Specification** - see Chapter I, page 1. Specifies the physical format of a CD-I disc, and how various information types (audio data, video data, and program related data) are coded on a CD-I disc.

**CD-I Physical Format** - see Chapter II, page 1. Defines how a CD-I disc is split into the different types of addressable physical sectors (and tracks), and the general encoding and decoding algorithms of these sectors.

## CD-I Full Functional Specification

### Appendix I

#### A I Glossary of Terms

---

**CD-I Physical Sector** - see Chapter III, page 1. A CD-I disc contains directly addressable sectors numbered consecutively beginning at zero. These are CD-I Physical sectors which contain 2352 bytes representing the synchronization, the header and the data fields.

**CD-I Physical Sector Formatter** - see Chapter II, page 35. A unit which superimposes the CD-I sector format on the incoming data.

**CD-I Sector** - see Chapter II, page 14. A unit of data of 2352 sequential bytes, directly addressable.

**CD-I Sector Data** - see Chapter II, page 37. Sectors containing information encoded in CD-I format.

**CD-I Sector Processor** - see Chapter II, page 38. Part of the CD-I decoder model which decodes the CD-I sectors.

**CD-I Standard Character Set** - see Chapter VI, page 7. Conforms to ISO 8859-1. The corresponding font module is always present in ROM in the Base Case System.

**CD-I System** - see Chapter I, page 1. A real-time decoder system capable of playing CD-I discs.

**CD-I System Validation** see Chapter VIII, page 22. The procedure for checking the correct operation of a CD-I system according to the CD-I Full Functional Specification and using a validation disc.

**CD-I Track** - see Chapter II, page 15. A data track containing only mode 2 sectors conforming to the CD-I specification.

**CD-RTOS** - see Chapter VII. Compact Disc Real Time Operating System: The name of the operating system used in CD-I players.

**CD-RTOS I/O Service Request** - see Chapter VII, page 3. A service request from a process to perform a device independent input/output function.

**CD-RTOS Kernel** - see Chapter VII, page 1. The nucleus of CD-RTOS which is responsible for service request processing, memory management, system initialization, multi-tasking, input/output management and exception and interrupt processing.

**CD-RTOS System State Service Request** - see Chapter VII, page 4. A request made to the CD-RTOS kernel that is only allowed for processes that are running in system state.

## Appendix I

### A I Glossary of Terms

---

**CD-RTOS User State Service Request** - see Chapter VII, page 1. A request made to the CD-RTOS kernel by a process that is running in either user state or system state.

**Channel Number** - see Chapter II, page 23. A number assigned to pieces of information contained in CD-I sectors to facilitate the selection of such information.

**Character Output Functions** - see Chapter VII, pages 190 to 195. User Communications Manager functions which are used to display text on the screen.

**Check Polynomial** - see Chapter II, page 27. A parameter used in detecting errors in a string of data.

**CIRC** - see Chapter II, page 26. Cross-interleaved Reed Solomon code, a technique used in error correction.

**CLUT** - see Chapter V, page 30. See Color Look-Up Table.

**CLUT Animation** - see Chapter V, page 72. For CLUT coded images, the 256 values in the CLUT control the colors of the entire image. Therefore, some simple animation effects are possible simply by redefining some or all of the CLUT contents as a function of time.

**CLUT (4,7 and 8-bit and Run-length Coded 3 & 7-bit)** - see Chapter V, pages 48-51. A series of standard representations of video signals defined for CD-I.

**Coded Character Set** (File Structure Volume Descriptor) - see Chapter III, page 7. A method of coding an agreed alphabet of letters, numbers and other symbols.

**Coded Character Set Identifier** - see Chapter III, page 9. Identifies a character set according to the International Register of Character Sets.

**Coding Information** - see Chapter II, page 23. A byte in the subheader that defines the details of the information located in the data area of the sector.

**Color Key** - see Chapter V, page 76. A color look-up table image may be compared on a RGB basis, pixel-by-pixel to a true color key, value K. The value of the comparison (true or false) may be used to effect the image transparency.

**Color Look-Up Table** - see Chapter V, page 30. A table containing all the colors which may be used in a particular picture. Each entry provides an RGB value. The picture may then be encoded using the table entry addresses rather than the direct RGB values.

## Appendix I

### A I Glossary of Terms

---

**Compact Disc File Manager** - see Chapter VII, page 14. A CD-RTOS file manager which handles I/O requests for the compact disc drive and the audio decoder. Provides random access to disc files at the byte level through system calls.

**Compact Disc Real Time Operating System** - see Chapter VII, page 1. The name of the operating system used in CD-I players.

**Compatibility** - see Chapter V, page 16. The property mandatory for CD-I applications, that they work correctly on all CD-I decoders, in the following respects: all essential information is presented to the user, and the user has access to all functions of the application.

**Compatible Format** - see Chapter V, page 14. A coding format for CD-I images which has equal and opposite minimum aspect ratio distortion for the '525 line monitor' and '625 line' display formats.

**Compression** - see Chapter V, page 18. A technique in which the amount of information used to present a specific image is reduced by eliminating redundant or unnecessary information.

**Computer Graphics** - see Chapter V, page 18. Pictures created by computer programs. Standard drawing functions such as 'line', 'circle', etc. are normally used.

**Concurrent (audio) channel** - see Chapter IV, page 28. Block multiplexed audio information. CD-I allows for up to 16 audio channels to be recorded concurrently on the disc, so that by playing the disc several times and accessing different channels, extended playing time is obtained or by playing the disc once, the same or related audio information may be obtained from parallel channels, e.g. the same information in a different language.

**Configuration Status Descriptor** - see Chapter VII, page 8. A list of Device Status Descriptors that describes the hardware configuration of a particular CD-I system.

**Control Field** (of a Q-channel) - see Chapter II, page 8. A 4-bit field immediately following the subcoding "sync patterns" within any subcoding block of the Q-channel. (See page 41 of the CD-DA specification).

**Coordinate Transformation** - see Chapter VII, page 83. A software mechanism in the User Communications Manager which provides resolution independence for User Communications Manager functions. For example, a circle of a specific radius drawn in a drawmap will appear to be the same on the display screen whether it is drawn in a normal, double or high resolution drawmap.

Appendix I

A I Glossary of Terms

---

**Copyright File Name** - see Chapter III, page 9. Field of the File Structure Volume Descriptor identifying the path name of a file used to produce a copyright message when the disc is first mounted.

**CRC** - see Chapter II, page 27. See Cyclic Redundancy Check.

**Creation Date and Time** - see Chapter III, page 10. Field of the File Structure Volume Descriptor which contains the date and time when the CD-I disc was originally mastered.

**Cursor Plane** - see Chapter V, page 91. A small graphical image plane that can be moved around the display or made invisible as required. It can be positioned at any position over the other planes.

**Cut** - see Chapter V, page 66. A visual effect in which an image is caused to appear suddenly, usually to replace a previous image.

**Cyclic Redundancy Check (CRC)** - see Chapter II, page 27. An error detection code scheme for the CD-I sectors data field.

Appendix I

A I Glossary of Terms

---

**Data Integrity** - see Chapter II, page 26. The preservation, against loss or corruption, of programs or data for their intended purposes.

**Data Preparer Identifier** - see Chapter III, page 9. Field of the File Structure Volume Descriptor identifying the person or other entity responsible for the preparation of data recorded on the volume.

**Data Retrieval Structure** - see Chapter III. The organisation of the storing of and searching for large quantities of data and making selected data available.

**Data Symbol** - see Chapter II, page 2. An entity made up of  $n$  bits representing a value between 0 and  $2^{n-1}$ .

**Data Track** - see Chapter II, page 3. A CD-I track with data encoded as 8-bit wide symbols (bytes) organized in sectors.

**DCP** - see Chapter VII, page 77. See Display Control Program.

**Decoder model** - see Chapter V, page 33. Representation of a method of translating or determining (decoding) encoded information.

**Delta YUV** - see Chapter V, page 19. "YUV" is a picture coding technique in which each pixel is represented by a measure of its luminance (brightness: Y) and chrominance (color: U and V) components. Delta YUV is a compression technique in which the differences between successive Y, U and V values are encoded rather than the absolute quantities.

**Derived Resolutions** - see Chapter V, page 6. Double and high resolution.

**Device Descriptor** - see Appendix VII.2, page 32. A data module consisting of a table of initial values of the parameters for a particular hardware device.

**Device Status Descriptor** - see Chapter VII, page 8. A descriptor that lists the capabilities of a device. The possible capabilities are specified, per device type, in Appendix VII.2.

**Directly Addressable Sector** - see Chapter III, page 1. A sector that may be accessed directly (as opposed to sequentially) through its address in terms of time and number of sectors.

**Directory Files** - see Chapter III, page 19. Files used extensively by the Compact Disc file management system to locate other files on a CD-I disc.



### Appendix I

#### A I Glossary of Terms

---

**Directory Record** - see Chapter III, page 19. A record describing one file in a directory.

**Directory Search Method** - see Chapter III, page 24. The method used to locate a specific file on the disc. This method allows any file to be opened using only one seek.

**Directory Structure** - see Chapter III, page 19. The organization of file descriptor records within a directory.

**Disc Directory** - see Chapter III, page 18. Directory structure of the CD-I disc starting with a Root directory and containing names and characteristics of files and subdirectories of the disc.

**Disc Label** see Chapter III, page 4. The information in the first track of a CD-I disc concerning the disc type and format, the status of the disc as a single entity or part of an album, the data size and the position of the file directory and boot modules.

**Disc Label Record Type** - see Chapter III, page 7. A byte in the File Structure Volume Descriptor Record that specifies whether the standard character set or an alternate character set is used for the text fields in the File Structure Volume Descriptor, directory records and path table entries.

**Disc Logical Format** - see Chapter III, page 1. Specifies the contents of CD-I sectors.

**Display Controller** - see Chapter VIII, page 2. Two-path device which takes pixel data from the two banks of RAM of a CD-I player and combines them to produce a single analog RGB video output.

**Display Control Program** - see Chapter VII, page 77. A set of command codes which are interpreted by the display hardware during either the horizontal or vertical retrace periods. The codes can be used to perform a variety of functions: definition of background colors, entries for color look-up tables, display parameters, etc.

**Display Line Start Pointers** - see Chapter V, page 57. Indicates where in memory the display shall start on a line-by-line basis. If there is no display line start point for a given line, the line will start at a memory address directly after the end of the previous line.

**Dissolve** - see Chapter V, page 81. The simultaneous fade-in of one image and fade-out of another.

### Appendix I

#### A I Glossary of Terms

---

**Double Resolution** - see Chapter V, page 6. A double resolution picture has twice as many pixels as a normal resolution picture in the horizontal direction, but the same number in the vertical direction.

**Double written** - see Chapter II, page 22. A method of improving data integrity whereby the data is written twice with a 4-byte separation. Achieves data integrity levels equivalent to Mode 1 EDC/ECC.

**Drawmap** - see Chapter VII, page 76. A block of memory allocated by the User Communications Manager to store image data.

**DSD** - see Device Status Descriptor.

**DYUV** - see Chapter V, page 19. See Delta YUV.

**Dynamic CLUT Update** - see Chapter V, page 73. CLUT entries can be reassigned with new RGB values via the display control program during the line or field retrace period. This technique allows the number of available colors to be extended beyond the respective maximum numbers of 8, 16, 128 and 256 for CLUT 3, CLUT 4, CLUT 7 and CLUT 8 encoded pictures.

Appendix I

A I Glossary of Terms

---

**ECC Codeword** - see Chapter II, page 28. Part of the error correction code field.

**ECC Field** see Error Correction Code field.

**EDC Codeword** - see Chapter II, page 27. A unit of information or codeword used in the error detection process.

**Error Detection Code Field** - see Chapter II, page 27. The field containing error detection code information.

**EDC Field** - see Chapter II, page 27. See Error Detection Code field.

**EDC Parity** - see Chapter II, page 27. Part of the information used in the error detection process.

**Effective Date and Time** - see Chapter III, page 10. If the data on the disc are not valid before a specific time, this field of the File Structure Volume Descriptor indicates the time at which the data become useful.

**Emphasis** - see Chapter IV, page 16. Increasing the level of higher signal frequencies relative to the lower frequencies prior to recording or broadcasting. Used to improve signal-to-noise ratio.

**Empty Sector** - see Chapter II, page 34. A Form 1 or Form 2 sector in which the data field consists of 2324 bytes which are all set to 0.

**Encoding model** - see Chapter V, page 19. Representation of a method of translating (encoding) from one method of expression to another.

**End-of-File bit** - see Chapter II, page 24. A bit in the submode byte of the subheader that is set to 1 in the last sector of a file.

**End-of-Record bit** - see Chapter II, page 25. A bit in the submode byte of the subheader that is set to 1 for the last sector of a real time or non-real time record.

**EOF-bit** - see Chapter II, page 24. see End-Of-File bit

**EOR-bit** - see Chapter II, page 25. See End-Of-Record bit

**Error Concealment** - see Chapter V, page 64. A variety of techniques used for concealing errors in visual images displayed from a CD-I disc.

Appendix I

A I Glossary of Terms

---

**Error Correction Code Field** - see Chapter II, page 28. The field containing error correction code information.

**Error Flag** - see Chapter II, page 39. Part of the data field in the error detection process which indicates whether an error has been detected in the given sector.

**Even/Odd Lines Flag** - see Chapter VII, page 97. When error concealment is used, this flag indicates whether the sector contains the even or odd lines of the image.

**Executable Object Code** - see Chapter VI, page 2. The output from a compiler, an assembler, a linkage editor or a linker, which is in the machine code for a particular processor, with each loadable program being one named file (module).

**Expiration Date and Time** - see Chapter III, page 10. If the data on the disc are valid only for a limited time, this field of the File Structure Volume Descriptor indicates the time at which the data become obsolete.

**Extended Attribute Record** - See Chapter III, page 20. A number of data blocks at the beginning of a file reserved for extended attribute information.

Appendix I

A I Glossary of Terms

---

**Fade** - see Chapter V, page 81. A gradual decrease (fade-out) or increase (fade-in) in the brightness level of an image.

**Field Control Table** - see Chapter V, page 54. A one-dimensional array of Display Control instructions which are carried out before the start of each field.

**File** - see Chapter III, page 28. A collection of logically related records identifiable in a directory

**File Descriptor Record** - see Chapter III, page 19. Used to store common format and attribute information needed to access a file on a disc.

**File Flags** - see Chapter III, page 21. Field of the File Descriptor Record containing information concerning the status of a file, e.g. hidden.

**File Manager** - see Chapter VII, page 11. A CD-RTOS module that processes raw data streams for use by a class of similar devices.

**File Number** - see Chapter II, page 23. Identifies all sectors that belong to one and the same file.

**File Pointer** - see Chapter VII, page 26. The location in a file where the next operation is to be performed.

**File Structure Standard Version Number** - see Chapter III, page 10. Defines the revision number of the file structure standard which the disc file structure obeys.

**File Structure Volume Descriptor** - see Chapter III, page 4. Record of the Disc Label describing all necessary items related to the files or parts of the files recorded on the volume.

**Filter and Range Selector** - see Chapter IV, page 16. Generates the values called filter (F) and range (R) depending on the peak values of the predictor units.

**First Order & Second Order Digital Filters** - see Chapter IV, page 14. Filters employing digital techniques, based on equations involving the first or second derivative respectively.

**Font Module** - see Chapter VII, page 86. A CD-RTOS data module that contains the bitmap images for each displayable character.

## CD-I Full Functional Specification

### Appendix I

#### A I Glossary of Terms

---

**Form** - see Chapter II, page 24. Information on a CD-I track may be recorded in either Form 1 or Form 2 format, depending on the level of data integrity required.

**Form bit** see Chapter II, page 24. Bit in the submode byte of the subheader field defining the data form (Form 1 or Form 2) for the sector.

**Form Value Detector** - see Chapter II, page 36. A unit which detects the value of the form bit.

**Frame** - see Chapter II, page 11. The frame is the basic entity for data addressing in CD-DA: a CD player reads 75 frames per second.

**Frame Number** - see Chapter II, page 11. Identification, for explanation purposes, of the successive frames.

Appendix I

A I Glossary of Terms

---

**Gain Control** see Chapter IV, page 14. Multiplication of the data by a gain factor.

**Getstat** - see Chapter VII, page 18. An I/O service request that is used to retrieve information about a specific device; typically a getstat function call will not be device independent.

**GF** - see Chapter II, page 28. See Goloia Field.

**Goloia Field** - see Chapter II, page 28. A finite field concept used in error correction coding.

**Glyph** - see Chapter VII, page 85. The bitmap image for a symbol. For example, the bitmap image of the letter "a" is the glyph for the letter "a".

**Glyph Data Table** - see Chapter VII, page 88. A table in a font module which contains information about each glyph in the font module.

**Granulation** - see Chapter V, page 89. An effect whereby the resolution of an image changes without its size altering. It is produced by a combination of pixel hold and line hold functions.

**Graphics Cursor Functions** - see Chapter VII, pages 111 to 117. User Communications Manager functions which control the shape, size, color and position of the graphics cursor on the display screen.

**Graphics Drawing Functions** - see Chapter VII, pages 139 to 158. User Communications Manager functions which are used to draw images in drawmaps.

**Group Execute (File attribute bit)** - see Chapter III, page 22. This bit of the attribute field of a directory record specifies that only users belonging to an identified group can execute the program contained in this file.

**Group Read (File attribute bit)** - see Chapter II, page 22. This bit of the attribute field of a directory record specifies that only users belonging to an identified group can access this file.

**Grouping Factor** - see Chapter V, page 100. To support error concealment by line repeat, the video data elements of the even and odd lines of an image are separated from each other and the two resultant data streams interleaved on disc. The interleaving breaks each data stream into groups of a constant number of sectors. This number is called the 'grouping factor'.

Appendix I

A I Glossary of Terms

---

**Hardware Configuration Status Descriptor** - see Chapter VII, page 8. See Configuration Status Descriptor.

**Header Field** - see Chapter II, page 21. A four-byte field, within a CD-I sector which contains the CD-I sector address and the mode information.

**High Resolution** - see Chapter V, page 6. A high resolution picture has twice as many pixels as a normal resolution picture in both the horizontal and vertical directions.



## Appendix I

### A I Glossary of Terms

---

**Image Coding Method** - see Chapter V, page 18. One of a number of ways of coding the RGB pixel values of an image, usually with the aim of compressing the amount of data needed to represent the image. The specified methods are RGB555, DYUV, QHY, CLUT and Runlength CLUT.

**Image Contribution Factor** - see Chapter V, page 80. See 'Image Mixing'.

**Image Line Pointer Table** - see Chapter V, page 57. Each line of an image as stored in image memory may start at an arbitrary address within the available image memory. An image line pointer table of line start addresses may be maintained for each image.

**Image Mixing** - see Chapter V, page 81. The images of two superimposed planes, A and B, may be intermixed after decoding to RGB on a pixel basis. The resultant image is equal to the sum of the two images, each multiplied by its own Image Contribution Factor.

**Image Plane** - see Chapter V, page 39. A displayed image may be formed by the superimposition of a number of component pictures. Each of these constitutes an image plane.

**Index** see Chapter II, page 10. Subdivision of a track in the program area only. Expressed in 2-digit BCD in the Q-channel of the Compact Disc subcode.

**Information Area** - see Chapter II, page 10. Synonymous to recorded area. Contains the lead-in area, program area and lead-out area.

**Information Track** - see Chapter II, page 3. One of the up to 99 tracks on a Compact Disc containing encoded information. Each track has a minimum length of 4 seconds and can either be a data or an audio track.

**Information Types** - see Chapter I, page 1. Three types of information are identified in CD-I namely audio data, video data and program related data.

**Init Module** - see Chapter VII, page 6. The system initialization module that contains the parameters to be used in starting up the system.

**Interlace** - see Chapter V, page 38. In television and computer graphics, a system of scanning a picture using two fields. The first line-by-line scan (field) sweeps alternate line positions on the picture, the second sweeps the gaps between the first, completing the total structure of the picture. Interlace scanning reduces the flicker inherent with low refresh rates.

## Appendix I

### A I Glossary of Terms

---

**Interleaved Files** - see Chapter III, page 31. Interleaved blocks of data from one file with blocks from other files on the CD-I disc. Files having their sectors physically interleaved on disc.

**Interleaving (sectors)** - see Chapter III, page 31. The process of physically separating the sectors within one logical file so that the data can be retrieved at the rate required for processing. The audio blocks in a real-time file must arrive from a CD-I disc at the audio processor at a very precise rate. Sectors between successive audio sectors can be used in a variety of ways: video or text data sectors can be placed within these gaps.

**International Reference Version ISO 646 Character Set** - see Chapter III, page 7. Identifies the character set described in the International Standard ISO 646 as International Reference Version.

**Interpolation** - see Chapter V, page 42. The process of filling in intermediate values of a series between known values of the series, by computation from the known values.

**Interpolation Scheme** - see Chapter II, page 22. A method of concealing errors.

**Irregular Updates** - see Chapter V, page 70. One of the two types of partial updates available within CD-I, where the area to be updated is irregular in shape.

**ISO 646** - see Chapter III, page 7. The international standard for a 7-bit coded character

**ISO 2022** - see Chapter III, page 7. International Standard specifying the methods (through shift function or escape sequences) to extend 7 or 8 bit coded character sets.

**ISO 2375** - see Chapter III, page 8. International Standard specifying the registration procedure of escape sequences used for code extension of a character set. It refers particularly to the "International Register of coded character sets to be used with escape sequences".

**ISO 8859-1** - see Chapter VI, page 7. The specification of the 8-bit coded character set for the Latin Alphabet Number 1, used for the CD-I standard character set definition.

## CD-I Full Functional Specification

### Appendix I

#### A I Glossary of Terms

---

**Kernel** - see Chapter VII, page 1. The heart of an operating system that is responsible for most system-related functions.

**Keyboard Input Functions** - see Chapter VII, pages 205 to 211. User Communications manager functions which are used to get character data from a keyboard device.

Appendix I

A I Glossary of Terms

---

**LBN** - see Chapter III, page 20. See Logical Block Number.

**Lead-in area** see Chapter II, page 4. A track (number 0) preceding the program area of a Compact Disc.

**Lead-in Q-channel Frame** - see Chapter II, page 8. A frame of information contained in the Q-subchannel, in the lead-in area of a Compact Disc: the Table of Contents of the CD-DA tracks is contained in such frames.

**Lead-out area** see Chapter II, page 7. A track (number \$AA) following the program area of a Compact Disc.

**Line Control Table** - see Chapter V, page 54. A two-dimensional array of Display Control instructions, each row of which is associated with a displayed line.

**Line Hold, Line Repeat** - see Chapter V, page 88. In the vertical direction, mosaic effects are produced by repeating lines using the display line start pointers in the DCP.

**Line Pointer Table** - see Chapter V, page 57. See Image Line Pointer Table.

**Linear Interpolation** - see Chapter V, pages 28 and 42. Interpolation in which an intermediate value between two known values is taken as the average of these two values. Used in the process of encoding QHY data to obtain high resolution images from normal resolution images and in the decoding of the U and V components of a DYUV image.

**Loadable Program** - see Chapter VI, page 2. A named file containing object code information used in CD-I.

**Logical Block Number (LBN)** - see Chapter III, page 20. The block address of an element of the disc file structure.

**Logical Block Size** - see Chapter III, page 9. Defines the size of a block in bytes as seen by the file system. For CD-I this is always 2048 bytes.

### Appendix I

#### A I Glossary of Terms

---

**Magnification** - see Chapter V, page 89. The pixel and line repeat functions may be used to produce an image magnification effect, e.g. for examining details of a structure of the image.

**Main Channel Address** - see Chapter III, page 1. See Absolute Disc Address.

**Matte** - see Chapter V, page 82. Any connected area of the display. It is used to control the transparency of image planes.

**Memory Access Controller** - see Chapter VII, page 75. A hardware component that retrieves pixel data from memory and passes it to the video real-time decoder for display. The access to memory is shared with the processor.

**Memory Module** - see Chapter VII, page 11. A named block of executable program statements or data that can be loaded by CD-RTOS into memory.

**Message Sectors** see Chapter II, page 34. CD-I sectors containing an audio message encoded as CD-DA audio data. This message asks the user to lower the volume, if a CD-I track is being played on a simple CD-DA player.

**Mode 2** See Chapter II, page 4. One of the two physical sector formats defined for CD-ROM. CD-I sector specification further describes two types of "Mode 2" sectors according to their "submode" contents:

Form 1 sectors incorporate EDC/ECC error detection and correction fields.

Form 2 sectors incorporate an auxiliary data field instead of EDC/ECC error detection and correction fields.

**Mode byte** - see Chapter II, page 21. Byte in the sector header field defining the data mode. Always Mode 2 for CD-I tracks.

**Mode Detector** - see Chapter II, page 39. Part of one of the units in the CD-I decoder model which detects the mode value of a data sector.

**Modification Date and Time** - see Chapter III, page 10. The date and time field in the File Structure Volume Descriptor indicating when the disc was last changed, e.g. re-mastered.

**Mosaic Pixel Repeat Factor** - see Chapter VIII, page 11. Factor by which the resolution of an image is reduced through the mosaic effects (see V.5.11). Its value is restricted to 2, 4, 8 or 16 in the Base Case system.

Appendix I

A I Glossary of Terms

---

**Mosaics** - see Chapter V, page 86. A group of visual effect functions which have the effect of reducing an image's resolution, by repeating or holding pixel values. See Pixel Hold and Pixel Repeat.

**Mouse** - see Appendix VII.2, page 17. A palm sized unit equipped with, typically, two control buttons, used as an X-Y pointer device to manipulate a screen display and invoke and control utility functions. Functions are invoked by moving the mouse to designated areas and pressing one of the buttons.

**MPU** - see Chapter VI, page 1. Micro processing unit.

Appendix I

A I Glossary of Terms

---

**Natural Images** - See Chapter V, pages 18, 19. Pictures which are photographic in nature and appear realistic.

**Near-instantaneous Compression** - see Chapter IV, page 14. Compression performed quickly enough to have no perceptible effect on the timing of the presentation of the information concerned.

**Noise Shaper Unit** - see Chapter IV, page 16. Compensates for the extra noise introduced in the predictor unit.

**Non-Volatile Memory** - see Chapter VIII, page 23. A memory able to retain its contents when the main power to the unit is removed.

**Non-Volatile RAM File Manager** - see Chapter VII, page 11. The file manager that is used to maintain the contents of the non-volatile RAM in the CD-I system.

**Normal Resolution** - see Chapter V, page 5. A displayed image may be regarded as a rectangular array of pixels. The resolution refers to the number of pixels in the horizontal and vertical directions. A normal resolution picture is a picture with the lowest resolution defined in the CD-I system and reflects the resolution of present 525/625 color television sets.

**NRF** - see Chapter VII, page 256. See Non-Volatile RAM File Manager.

**Number of Volumes in Album** - see Chapter III, page 9. Total number of discs in the album to which a disc belongs.

**NVRAM** - see Chapter VIII, page 23. See Non-Volatile Memory

### Appendix I

#### A I Glossary of Terms

---

**Object Code** - see Chapter VI, page 2. The code of a user's program after it has been translated by means of an assembler or a compiler and a linker.

**Odd/Even Line Separation** - see Chapter V, page 63. When error concealment is to be used, the even and odd lines of an image are coded on a disc in separate sectors and are physically separated by interleaving so as to minimize the chance of an error occurring on adjacent lines.

**One dimensional Run-length Coding** - see Chapter V, page 18. A picture coding technique in which pixel data is compressed using run-length coding in the horizontal direction only.

**Overlay** - see Chapter V, page 75. The process of superimposing image planes in a given visual image.

**Overscan** - see Chapter V, page 5. The extension of the deflection of the electron beam of a Cathode Ray Tube (CRT) beyond the usable physical dimensions of the screen. In this way an image always fills the visible part of the display screen, but some of the image is lost at the edges.

**Owner Execute (File attribute bit)** - see Chapter III, page 22. This bit of the attribute field of a directory record, if set to one, specifies that only the user identified as "owner" of the file can execute the program contained in this file.

**Owner I.D.** - see Chapter III, page 22. Contains the user identification number of the creator of a file.



Appendix I

A I Glossary of Terms

---

**Panning** see Chapter IV, page 27. The distribution of a mono signal between left and right audio channels.

**Parent Directory** - see Chapter III, page 24. A directory which contains sub-directories.

**Partial Update** - see Chapter V, page 11. New image data written to part of an image which has already been loaded into a drawmap.

**Path Descriptor** - see Chapter VII, page 227. A data structure used to represent an open file; each open file is associated with a path descriptor.

**Path Table, Path Table Index** - see Chapter III, page 24. The table or index used for directory search. Describes the entire directory structure on a disc. Permits any file on disc to be opened using only one seek.

**Path Table Entry** - see Chapter III, page 24. Individual element of the Path Table which is used to locate each directory or sub directory on a disc.

**Path Table Size** - see Chapter III, page 9. A field in the File Structure Volume Descriptor which defines the size in bytes of the system path table.

**PCM** - See Chapter IV, page 21. See Pulse Code Modulation.

**Physically Interleave Files** - see Chapter III, page 31. Interleave blocks of data of one file with blocks from other files on the CD-I disc; for example the interleaving of audio blocks in a real-time file with other data blocks.

**Pipeman** - see Appendix VII.1, Chapter 10. The Pipe File Manager, one of the file managers of CD-RTOS that supports interprocess communication through the use of "pipes".

**Pixel** - see Chapter V, page 5. An image may be considered to be made up of a number of small units known as pixels or picture elements. A pixel is the smallest element of an image which can be manipulated or identified.

**Pixel Aspect Ratio** - see Chapter V, page 14. The ratio of the height of a pixel to its width.

**Pixel Data Formats** - see Chapter V, pages 102 to 107. The disc formats of all the defined coding methods for pixel data.

## Appendix I

### A I Glossary of Terms

---

**Pixel Hold** - see Chapter V, page 87. Method used to reduce resolution by holding the RGB value of every  $n^{\text{th}}$  pixel for  $n$  pixel periods. This function operates after the pixel codes have been decoded to RGB.

**Pixel Pairs** - see Chapter V, page 6. In 3-bit run-length coded images and 4-bit CLUT encoded pictures, two pixels are put together to make up one byte, and are then regarded for most purposes (e.g. the length of the run in run-length coded pictures) as a single pixel pair.

**Pixel Repeat** - see Chapter V, page 88. A visual effect function that repeats pixel values from an image memory to produce horizontal magnification. Used in RGB 555 and CLUT image decoding only.

**Plane** See Image Plane.

**Play Control Block** - see Chapter VII, page 44. A data structure that contains the necessary information concerning the data to be accessed on the disc for the current or next Real-Time Record Play function of the Compact Disc File Manager.

**Play Control List** - see Chapter VII, page 48. A structured list that controls the destination of the data.

**Player's Default Character Set** - see Chapter III, page 5. Set of characters tentatively selected by the CD-I player when a disc is initially accessed. May be different from the CD-I standard character set (for instance Shifted JIS Kanji for a Japanese player).

**POINT, PMIN, PSEC, PFRAME** - see Chapter II, page 8. During the lead-in track of a CD-I disc a Table of Contents is present with-in the subcode Q channel (see p.43 of the CD-DA Specification): PMIN, PSEC and PFRAME give the starting point (on the absolute time scale ATIME) of the CD-DA track number, pointed to by POINT.

**Pointer Input Functions** - see Chapter VII, page 200. User Communications Manager functions which are used to get coordinate information from a pointing device.

**Polygon** - see Chapter VII, page 148. A closed figure that is bounded by straight lines.

**Polyline** - see Chapter VII, page 143. A sequence of lines that are connected end to end sequentially but do not necessarily form a closed figure.

Appendix I

A I Glossary of Terms

---

**Predictor Selector Unit** - see Chapter IV, page 16. Sends the data chosen by the predictor unit to the gain control unit.

**Predictor Unit (filters)** - see Chapter IV, page 16. A device (filters) used in ADPCM encoding to achieve effective response to audio-frequency distribution fluctuations.

**Primitive Polynomial** - see Chapter II, page 28. A polynomial generating a field in the error correction code process.

**Privileged Instructions** - see Chapter VI, page 5. Instructions that can only be executed by the processor in the "system state".

**Process Scheduling** - see Chapter VII, page 1. A method of allowing each of a series of processes to obtain some execution time on the microprocessing unit, allowing each process to eventually finish.

**Program Area** see Chapter II, page 5. The area of a Compact Disc containing the program and consisting of a maximum of 99 audio or data tracks.

**Program Related Data** - see Chapter VI, page 1. Data to be read and processed by the CD-I application.

**Program-Related Data Representations** - see Chapter VI, page 1. Conventions applicable to data to be read and processed by the CD-I application.

**Publisher Identifier** - see Chapter III, page 9. Field of the File Structure Volume Descriptor identifying the person or entity responsible for the definition of the volume contents.

**Pulse Code Modulation** - see Chapter IV, page 21. A technique for recording analog information in digital form. The analog signal is sampled and the sampled value is quantized, i.e. represented by a fixed length binary number. This number is then transmitted as a corresponding set of pulses.

**P-Words** - see Chapter II, page 29. Column codewords in the parity check matrix.

Appendix I

A I Glossary of Terms

---

**Q-Words** - see Chapter II, page 29. Row codewords contained in the parity check matrix.

**QHY** - see Chapter V, page 25. See Quantized High Y.

**QHY Quantization Levels** - see Chapter V, page 28. The QHY difference values, which need to be added to the normal resolution quantities to generate a pseudo-high resolution image, are 8-bit quantities varying between 0 and 255. Eight values are chosen from these 256 possible quantities. Each difference value is then made equal to the nearest of these eight, and a three bit number is used to represent it. The eight chosen values are known as quantization levels.

**Quantized High Y** - see Chapter V, page 25. A coding technique used to reduce the quantity of data required to encode a high resolution type picture: a normal resolution DYUV image is recorded together with the data which needs to be added to it to turn the luminance (Y) of the picture to the equivalent of high resolution. The latter data further compressed is termed 'QHY'.

**Quantizer** - see Chapter V, page 28. A device or circuit which assigns fixed binary values to sampled analog signals in analog-to-digital conversion.

Appendix I

A I Glossary of Terms

---

**RAM** - see Chapter VIII, page 3. Random Access Memory. In computing (1) a memory chip used with microprocessors. Information can be both read from, and written into the memory but the contents are lost when the power supply is removed, (2) any form of storage in which the access time for any item of data is independent of the location of the data most recently obtained, e.g. immediate access store has a random access capability but magnetic disk does not.

**Raw Sector** - see Chapter VII, page 38. A sector of information that includes not only the data field but also the header, subheader and error detection and correction information, if any.

**Read Only Memory** - see Chapter VII, page 6. In computing, a storage device whose contents can only be changed by a particular user, by particular operating conditions or by a particular external process. Read only storage can include storage media where the writing action is inhibited by the Operating System or by some mechanical device, e.g. a tag on a diskette. The term ROM implies a storage device not designed to be modified by conventional write procedures and which is used to store permanent information in computers and microcomputers, e.g. the Operating System and BASIC interpreters are often supplied in ROM on microcomputers.

**Real-Time Behaviour** - see Chapter II, page 26. The behaviour of a system able to follow real time constraints (such as the rotational speed of a CD-I disc).

**Real-Time Decoder** - see Chapter V, page 34. A hardware component that is capable of converting data from one format to another in real time.

**Real-Time File** - see Chapter III, page 30. A file containing at least one real-time record.

**Real-Time Record** - see Chapter III, page 30. A logical record in a CD-I file containing audio, video, and/or computer data that must be retrieved from a CD-I disc at a precise rate.

**Real-Time Sector** - see Chapter II, page 24. Data that has to be processed without interrupting the real-time behaviour of the CD-I system.

**Record** - see Chapter III, page 19. The logical component of a file.

**Recorded Area** - see Chapter II, page 3. The total area of a Compact Disc including the lead-in area, the program area and the lead-out area.

### Appendix I

#### A I Glossary of Terms

---

**Rectangular Updates** - see Chapter V, page 70. One of the two types of partial updates available within CD-I, where the area to be updated is in rectangular form.

**Reduced Resolution** - see Chapter V, page 90. The magnification effect produced by pixel and line repeat may be used to produce a low resolution image allowing fast update from the disc of a larger than normal animated screen area. This is an alternative to animation, like run-length compression, which may be chosen for specific applications.

**Reed Solomon Codewords** - see Chapter II, page 29. Codewords used in the Reed Solomon encoding and decoding process.

**RGB** - see Chapter V, page 18. Every color can be represented as the sum of different proportions of the three primary colors, red (R), green (G) and blue (B). In RGB encoding, every pixel is represented by its R, G and B components. These are values which, on presentation to suitable digital to analogue converters will give the correct voltages required by the red, green and blue guns of a cathode ray tube to produce the color of the pixel on the display screen.

**RGB555** - see Chapter V, page 47. A method of coding visual images as 5 bits per red, green and blue component. See also RGB.

**Region** - see Chapter VII, page 83. A software mechanism in the User Communications Manager which is used to limit the area of drawing in a drawmap and to set up mattes.

**Reserved Field** (within Form 2 sectors) - see Chapter II, page 33. A field reserved for quality control during the CD-I disc production process.

**Revision Number** - see Chapter VII, page 7. A field in the CD-RTOS module header that specifies which version of a module this is.

**ROM** See Read Only Memory.

**Root Directory** - see Chapter III, page 18. The highest level directory contained on a disc. A root directory must reside on every CD-I disc.

**Rotational Latency** - see Chapter III, page 31. The time taken for the disc to rotate under the read head until the required sector or frame becomes available for reading.

**Run-length Coding** - see Chapter V, page 49. A picture data compression technique which uses two-byte codes. The first byte identifies color and the second byte tells the decoder how many consecutive pixels are to be of this color.

## Appendix I

### A I Glossary of Terms

---

**Safety Area** - see Chapter V, page 8. That part of an image (expressed in terms of pixels) that is guaranteed to be displayed despite all tolerance values that can occur in monitors and TV sets.

**Sample Rate Converter** - see Chapter IV, page 1. Unit which converts PCM encoded audio signal into a new PCM encoded audio signal with a different sampling frequency.

**Scheduling Algorithm** - see Chapter VII, page 15. The algorithm used by the kernel to determine which process is to be scheduled for execution next.

**Scrambling** - see Chapter II, page 19. A technique used in CD-I. All data, except for the data in the synchronization field, is scrambled: the contents of a 15-bit shift register scrambler is EXOR-ed with the serial information bit by bit.

**Scramble Register** - see Chapter II, page 19. A register used in the scrambling and descrambling process for all data in a CD-I sector (except the synchronization field).

**Scroll** - see Chapter V, page 67. The repeated repositioning of a displayed image within a large image. Motion may be vertical, horizontal or a combination of the two.

**Sector** - see Chapter III, page 1. Reserved abbreviation used for a CD-I physical sector.

**Sector Address** - see Chapter II, page 21. The physical address of a sector expressed in minutes, seconds and sector number. Contained in the address part of the sector header.

**Sector Format Multiplexer** - see Chapter II, page 36. A unit which multiplexes the various sector information.

**Sector Interleaver** - see Chapter II, page 36. A unit which interleaves the sectors with various information types together.

**Sector Structure** - see Chapter II, page 26. The method of encoding data within a sector.

**Seek** - see Chapter VII, page 14. The action of changing, within a CD-I file, the location of the current file pointer. It also causes the CD drive reading head to move to the required location.

## Appendix I

### A I Glossary of Terms

---

**Service Request** - see Chapter VII, page 3. A request made to the kernel by an application to perform a specific activity.

**Setstat** - see Chapter VII, page 25. An I/O service request that is used to make a request or to change the status of a specific device; typically a setstat function call will not be device independent.

**Shifted JIS Kanji** - see Chapter VI, page 9. A Japanese standard defining a coded Kanji character set.

**Sleep** - see Chapter VII, page 4. A process execution state where the process will be inactive until a specific amount of time has passed or an interrupt is received.

**Sound Group** - see Chapter IV, page 9. Part of an audio block which contains four sound units, each consisting of four identical sound parameter bytes and 28 data bytes.

**Soundmap** - see Chapter IV, page 23. A memory area allocated by UCM for storage of ADPCM audio data.

**Soundmap Control Functions** - see Chapter VII, page 58. User Communications Manager functions which affect soundmaps.

**Sound Parameters** - see Chapter IV, page 9. Filter and range values describing the characteristics of a sound group.

**Sound Quality Levels** - see Chapter IV, page 3. Four levels of audio quality are defined in the CD-I specification. These are CD-Digital Audio with a bandwidth of 20 kHz and 16-bit quantization, CD-I Level A, 17 kHz bandwidth with 8-bit quantization, Level B, 17 kHz bandwidth with 4-bit quantization and Level C, 8.5 kHz bandwidth with 4-bit quantization.

**Sound Unit** - see Chapter IV, pages 9, 11. A unit consisting of four identical sound parameters and 28 sound data bytes.

**Source Image** - see Chapter V, page 19. High resolution, 8-bit PCM RGB image which is used as the source data before compression.

**Spatial Correlation** - see Chapter V, page 63. A characteristic of visual images where there is a high degree of similarity between two adjacent lines or pixels within an image.

**Standard Files** - see Chapter III, page 29. Files in the CD-I disc root directory with a recommended name: Path Table, Album Descriptor, Author and Manufacturer identification.



## Appendix I

### A I Glossary of Terms

---

**Start Flag** See Chapter II, page 8. A flag in the P-channel indicating the start of a track.

**Start Position** (of a track) see Chapter II, page 8. Position defined on the absolute time (ATIME) scale of the CD subcode, at which a track number changes and a new track starts.

**Subcode Channel** - See Chapter II, page 8. Besides the "main" data channel, 8 different Subcode Channels, called P, Q, .... W, are defined (see page 39 of the CD-DA specification). P and Q channels are used for addressing control, on disc, of the CD player itself (head positioning, etc.).

**Subheader** - see Chapter II, page 22. The subheader field is contained in each sector and defines file number, channel number, submode and coding information.

**Submode** - see Chapter II, page 23. The submode byte defines global attributes of the sector, for instance the nature of the data (audio, video, ...) it contains.

**Subsampling** - see Chapter V, page 22. The reduction of the resolution of a digitized image by (effectively) reconstituting the image in analog form and then resampling it at a lower rate. To avoid aliasing in the resulting image, anti-alias filtering must be employed (see Anti-aliasing Filter).

**Subscreen** - see Chapter V, page 13. A horizontal strip of an image plane extending the full width of the display screen, which may contain its own image of a different coding method to other subscreens simultaneously present. Subscreens are produced using the Display Control Program. (See Display Control Program).

**Synchronization Detector** - see Chapter II, page 39. Part of one of the units in the CD-I decoder model used for all timing needed in the sector processor.

**Synchronization to Display Scanning** - see Chapter V, page 74. If an object is repeatedly being drawn on the screen, erased and then drawn in a new position, such as in some computer graphic applications, a flicker effect is frequently seen. This effect should be avoided by synchronizing the writing of objects to the field scan so that erasure and re-writing takes place between successive scans of the object.

## Appendix I

### A I Glossary of Terms

---

**Synchronization Field** - see Chapter II, page 15. A 12 byte field at the beginning of a CD-I sector.

**Synchronization Primitives** - see Chapter VII, page 266. Low level mechanisms for use by application software to perform the required synchronization between the video, audio and data information.

**Synchronous Execution Service Request** - see Chapter VII, page 15. A service request for which the process will stop executing until the service request has been satisfied.

**System Identifier** - see Chapter III, page 8. Field of the File Structure Volume Descriptor identifying the operating system, (e.g. CD-RTOS) to be used to access the volume.

**System Modules** see Chapter VII, page 8. CD-RTOS modules that are required for the CD-I system to function. These include the kernel, file managers and device drivers.

**System Services** - see Chapter VII, page 1. Any of the variety of facilities provided by the "system" (i.e. CD-RTOS) including creation and termination of processes, allocation and deallocation of memory and processing of I/O requests.

**System State** - see Chapter VI, page 5. A special state of the processor which allows execution of privileged instructions; synonymous with "Supervisor" state for the M68000 family of microprocessors. File Managers and device drivers always execute in system state.

### Appendix I

#### A I Glossary of Terms

---

**Terminator, Terminator Record** - see Chapter III, page 4. The last record in the disc label, signifying the end of the disc label.

**TNO** - see Chapter II, page 10. Track number expressed in 2-digit BCD in the Q-channel of the Compact Disc subcode (see page 42 of the CD-DA specification).

**TOC** - see Chapter II, page 8. Table Of Contents: the sub-code information in the lead-in area (of a Compact Disc) identifying the number of tracks, and indicating their timing, duration and mode.

**Track** - see Chapter II, page 3. A sequence of contiguous data, the beginning, length, mode and end of which are defined in the table of contents, which is held in the Q subcode channel of the lead-in area of the disc. The two types of tracks currently defined are the CD-DA track according to the CD-DA specification and the data track according to the CD-ROM specification which is also used in CD-I. In CD-DA the length of a track is related to playing times between 4 seconds and 72 minutes.

**Transition Area** - see Chapter V, page 71. Relates to the partial updates of DYUV images. Here it is necessary to begin each line of a partial update with a transition area. This is a short sequence of DYUV codes which have been pre-calculated to make the transition from the YUV values in the surrounding image just preceding the update to those at the left edge of the update proper. Each line of a partial update must similarly be terminated with a transition area.

**Transparency Control** - see Chapter V, page 78. The three mechanisms used to control the transparency of superimposed image planes.

**Trap Handler** - see Chapter VII, page 3. A CD-RTOS module that typically contains a set of related subroutines that is linked to by an application at runtime rather than being included as part of the application module.

**Trigger Bit** - see Chapter II, page 24. A bit in the submode byte of the subheader that is interpreted by an application to cause synchronization of various events.

## Appendix I

### A I Glossary of Terms

---

**U** - see Chapter V, page 19. One of the two chrominance components of a video signal containing color information.

**User Communications Manager** - see Chapter VII, page 73. The CD-RTOS file manager which provides the software interface to the user input devices and to the video processor on a CD-I player.

**User State** - see Chapter VII, page 1. The state of the machine when user processes execute and user related requests can be processed.

Appendix I

A I Glossary of Terms

---

**V** - see Chapter V, page 19. One of the two chrominance components of a video signal containing color information.

**VDSQ** - see Chapter V, page 95. See Video Data Sequence.

**Video Data Sequence** - see Chapter V, page 95. The logical unit of image data. It typically contains the pixel data for a whole image or a partial update to an image.

**Video Decoder Delay** - see Chapter VIII, page 11. The time delay caused by the access controller and video decoder between enabling an image for display and the image appearing on the display itself. It should not be more than one field.

**Video Inquiry Functions** - see Chapter VII, page 181. User Communications Manager functions which return information related to the video device. For instance, applications can obtain the data of a font or the location of a region, using these functions.

**Video Real-Time Data Representation** - see Chapter V. The way that different kinds of visual images may be recorded on the disc, and the way that this data are to be interpreted in real time by the CD-I player.

**Video Sector** - see Chapter II, page 25. A CD-I sector in which the data field contains video data.

**Visual Coding Information Byte** - see Chapter V, page 96. Contains details of the coding method and resolution of each video data sequence. The video coding information byte is contained in the subheader of each sector of the video sequence.

**Volume Identifier** - see Chapter III, page 8. Field of the File Structure Volume Descriptor identifying the name of the CD-I disc (volume)

**Volume Structure Standard Identifier** - see Chapter III, page 8. This field of the File Structure Volume Descriptor represents the standard to which the disc file structure conforms (i.e. CD-I).

**Volume Structure Version Number** - see Chapter III, page 8. Defines the version number of the standard to which the disc conforms.

### Appendix I

#### A I Glossary of Terms

---

**Wipe** - see Chapter V, page 79. The replacement of one image by another during a period of time by the motion of a boundary separating the visible parts of the two images.

**Working Directory** - see Chapter VII, page 57. The directory currently used to find data files.

**World Execute (File attribute bit)** - see Chapter III, page 22. This bit of the attribute field of a directory record, if set to one, specifies that any user can execute this file.

**World Read (File attribute bit)** - see Chapter III, page 22. This bit of the attribute field of a directory record, if set to one, specifies that any user can access this file.

Appendix I

A I Glossary of Terms

---

**Y Component** - see Chapter V, page 19. The luminance or brightness component of a video signal.

**Zoom** - see Chapter V, page 89. In video and photography the facility to enlarge (zoom-in), or diminish (zoom-out), the area of interest in an image.

**2's complement** - see Chapter II, page 3. A method of binary notation in which binary numbers are negated by interchanging the one's and zero's and adding one to the result.

**625/525 Line Image Interchange** - see Chapter V, page 65. The possibility to display native 525-line images on 625-line displays and vice versa.

## CD-I Full Functional Specification

### Appendix I

---

This page is intentionally left blank



Table of Contents

---

**A II Subheader Values**

	<b>Page</b>
1 File Number	A II-1
2 Channel Number	A II-2
3 Submode	A II-4
4 Coding Information	A II-6
4.1 Audio	A II-6
4.2 Video	A II-7
4.3 Data	A II-8

This page is intentionally left blank

List of Illustrations

---

<b>Fig. No.</b>	<b>Caption</b>	<b>Page</b>
A II.1	File Number Byte Value	A II-1
A II.2	Channel Selection Register	A II-3
A II.3	Channel Number Allocations	A II-3
A II.4	Submode Byte Format	A II-4
A II.5	Allowed Combinations A/V/Data	A II-5
A II.6	Audio Coding Information Byte Format	A II-6
A II.7	Video Coding Information Byte Format	A II-7
A II.8	Data Coding Information Byte Format	A II-8

This page is intentionally left blank

A II Subheader Values

---

**A II Subheader Values****1 File Number**

The file number is used to identify all sectors that belong to one file. A given file may be physically interleaved with other files on the disc. All sectors of a logical file have the same value in the file number byte. The file number can be used to select sectors that belong to the same logical file and to reject any others. The file number with a byte value of 0 is only used for a file that will be read consecutively. No interleaving is possible with file 0. Files 1 to 255 (see Figure A II.1) may be read consecutively or may be interleaved with other files.

Figure A II.1 **File Number Byte Values**

Byte Value	File Number
0000 0000	0 (consecutive sector reading only)
0000 0001	1
0000 0010	2
:	:
1111 1111	255

---

## A II Subheader Values

---

### 2 Channel Number

A real-time record<sup>1</sup> may contain several information channels. Two notable reasons why this is allowed are:

- to minimize the amount of disc space wasted by gaps between audio blocks, and
- to allow one of several mutually exclusive sequences to accompany each other, e.g. one of a set of audio sequences with a video sequence.

A real-time record is created by merging several different pieces of information together. When this is done, each logical segment may be given a unique channel number, which is recorded in the second and sixth bytes of the subheader. Each real-time record has a maximum of 32 channels (channel numbers 0-31). Because the maximum number of audio channels is 16, audio sectors can only use channel numbers 0 to 15. In order to conserve channel codes and simplify programming, a common channel number can be used for sectors that are always to be selected together, regardless of the type of coding information (audio, video, etc.) contained in the sector.

Upon playback, any combination of channels in the record may be played together if such a combination is allowed in the Base Case System. The selection is controlled by a 32-bit channel selection register. Each bit in the register corresponds to one subheader channel number. If a bit is set to one in the channel selection register, the sectors for the corresponding channel number are selected. If the bit is reset, i.e. to zero, sectors for the corresponding channel are ignored. The assignment of bits in the channel selection register is illustrated in Figure A II.2.

---

<sup>1</sup> A real-time record is the smallest unit of real-time data that can be randomly accessed. Once a real-time record is invoked it must be played from the beginning to end in an uninterrupted way.

<sup>2</sup> Real-time data is data whose:

- real-time bit is set in the submode byte (see A II.3),
- flow from a CD-I disc through any interface cannot be interrupted within a real-time record if an error occurs or is detected at any stage of the I/O process,
- flow at each stage of the input to output process is determined by the disc data flow of 75 sectors/second.

## A II Subheader Values

Figure A II.2 Channel Selection Register

31	23	15	0
1000 0000	0000 0001	0001 0000	0000 0001

For the channel selection register depicted in Figure A.II.2, sectors would be selected if their subheader contained channel number 0, 12, 16, or 31.

Real-time records are always played back asynchronously. This means that the application program requesting playback continues to execute concurrently. It is possible for the application to change the channel selection register while playback is in progress, allowing an instantaneous switch between channels in real-time.

Non-real-time records can use any channel number (0 - 31).

Figure A II.3 Channel Number Allocations

Channel Byte	Channel Number	Data		
		A	V	D
0000 0000	0	A	V	D
0000 0000	1	A	V	D
:	:	:	:	:
0000 1111	15	A	V	D
0001 0000	16	-	V	D
:	:	:	:	:
0001 1111	31	-	V	D
0010 0000	reserved	-	-	-
:	:	:	:	:
1111 1111	reserved	-	-	-

Figure A II.3 gives which channel numbers may be used.

Channel numbers can be used for audio (A), video (V) and program related data (D) sectors, but audio channel numbers must be within the range of 0 to 15.

The audio channel selection for the ADPCM decoder is controlled by a separate 16-bit audio channel selection register. Any audio channels selected by the 32-bit register but not the 16-bit register are transferred to the host system along with the other (non-audio) channels; the audio channel selected by the 16-bit register is transferred to the audio processor.

A II Subheader Values

---

**3 Submode** (see also II 4.5.3)

The submode byte is bit-encoded as shown in Figure A II.4.

Figure A II.4 **Submode Byte Format**

Bit Number	Bit Name
7	End of File (EOF)
6	Real-Time Sector (RT)
5	Form (F)
4	Trigger (T)
3	Data (D)
2	Audio (A)
1	Video (V)
0	End Of Record (EOR)

- End Of File (EOF)** : The last sector of a file is indicated by bit 7 equal to 1. All other sectors have bit 7 equal to zero.
- Real-Time Sector** : If bit 6 has the value 1 then the data (RT) has to be processed without interrupting the real-time behavior of the CD-I system. For example, audio sectors have to be transferred to the ADPCM decoder in real time in order to avoid the overflow or underflow of data.
- Form (F)** : This bit has a value of 0 for all sectors recorded in Form 1 and a value of 1 for all sectors recorded in Form 2 (see IV, V and VI).
- Trigger (T)** : This bit is used to synchronize the application with various coding information, like visuals to audio, in real time. The bit when set to one generates an interrupt (see VII.4).
- Data (D)** : This bit has a value of 1 for program related data sectors. Otherwise it is zero. When this bit is set to one, the Form bit must be zero.
- Audio (A)** : This bit has a value of 1 for audio sectors. Otherwise it is zero. When this bit is set to one the Form bit is also set to one.



## A II Subheader Values

Video (V) : This bit has value of 1 for video sectors. Otherwise it is zero.

End Of Record (EOR) : This bit must have the value 1 for the last sector of a logical record. Other-wise it is zero. The use of the EOR bit is only mandatory for real-time records.

The allowable combinations of the form, audio, video and data bits are specified in Figure A II.5.

Figure A II.5 **Allowed Combinations A/V/Data**

Coding (bits 3,2,1)	Form 1 (Form bit = 0)	Form 2 (Form bit = 1)
Audio = 1	NA	OK
Video = 1	OK [NRT]	OK
Data = 1	OK	NA

where

OK = allowed combination

NA = not allowed combination

NRT = only to be used for non-real-time sectors

Only one of bit 1, bit 2 or bit 3 may have the value 1 at the same time. One of bit 1, bit 2 and bit 3 must be set to the value 1 for all sectors except empty and message sectors. If bits 1, 2 and 3 of the submode byte are zero then the sectors are either empty or message sectors.

An audio sector must contain 2304 bytes and therefore cannot be used in Form 1. A data sector cannot be used in Form 2 because data errors cannot be concealed and the integrity of the data cannot be guaranteed.

Since Form 1 real time sectors may not be corrected in case of error, if this would endanger the real time behaviour of the system, it is recommended:

- to group all the Form 1 sectors that need to be corrected in a separate record placed just before the real time record and to declare these Form 1 sectors as "non real time" (RT = 0)
- not to mix, in the same record, real time and non real time sectors.

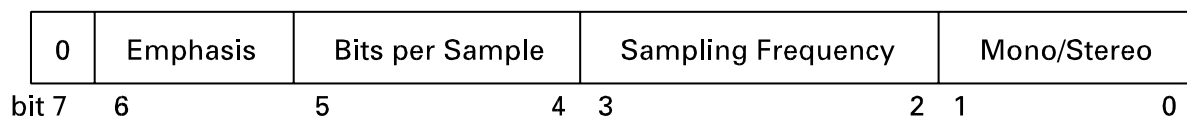
A II Subheader Values

---

**4 Coding Information****4.1 Audio**

The audio Coding Information byte is defined as follows:

Figure A II.6 **Audio Coding Information Byte Format**

**Reserved**

Bit 7 = Zero

**Emphasis (see IV.4.2)**

Bit 6

- 0 = Emphasis off
- 1 = Emphasis on

**Number of Bits per Sample**

Bits 5-4

- 00 = 4 bits
- 01 = 8 bits
- 10 = Reserved
- 11 = Reserved

**Sampling Frequency**

Bits 3-2

- 00 = 37.8 kHz
- 01 = 18.9 kHz
- 10 = Reserved
- 11 = Reserved

**Mono/Stereo**

Bits 1-0

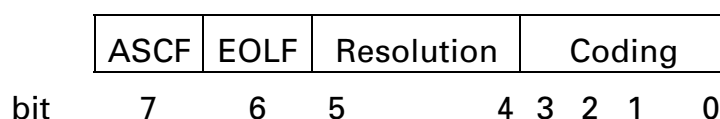
- 00 = Mono
- 01 = Stereo
- 10 = Reserved
- 11 = Reserved

## A II Subheader Values

## 4.2 Video

Details of the coding method and resolution of each video data sector are contained in the Video Coding Information Byte in the subheader of each video sector. The format of this byte is:

Figure A II.7 Video Coding Information Byte Format



where

**ASCF = Application Specific Coding Flag**

Bit 7

- 0 = coding conforms to the specification of chapter V
- 1 = coding is application-specific, i.e. to be interpreted by application software.

**EOLF = Even/Odd Line**

Bit 6

- 0 = Even lines
- 1 = Odd lines

If error concealment is to be used, this flag indicates whether the sector contains the even or the odd lines of the image. If error concealment is not to be used, this bit is set to zero.

**Resolution**

Bits 5-4

- 00 = Normal
- 01 = Double
- 10 = Reserved
- 11 = High

**Coding**

Bits 3 2 1 0

- 0000 = CLUT4            1000 QHY
- 0001 = CLUT7           1001 to 1111 Reserved
- 0010 = CLUT8
- 0011 = RL3
- 0100 = RL7
- 0101 = DYUV
- 0110 = RGB555 (lower)
- 0111 = RGB555 (upper)

A II Subheader Values

---

**4.3 Data**

The data or program related data Coding Information byte is defined as follows:

Figure A II.8 **Data Coding Information Byte Format**

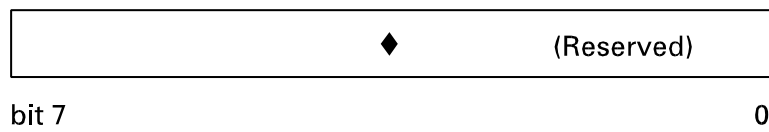


Table of Contents

---

	<b>Page</b>
A V.1 <b>CCIR Level Assignment</b>	A V-1
A V.2 <b>Guidelines for 525/625 Image Interchange</b>	A V-3
2.1    Introduction	A V-3
2.2    Image Interchange	A V-4
2.2.1    Display Compatibility Mode	A V-4
2.2.2    Fitting the Image to the Display	A V-6
2.2.3    The LCT and Visual Effects	A V-8
2.2.4    Cursor Alignment	A V-9
2.2.5    Pointer Alignment	A V-10
2.2.6    More than One Standard on a Disc	A V-11
2.3    Field Timing	A V-12
2.3.1    Motion Video	A V-13
2.3.2    Object Motion	A V-14

This page is intentionally left blank

## CD-I Full Functional Specification

### Appendix V

### Video Real-Time Data Representation

#### List of Illustrations

---

<b>Fig. No.</b>	<b>Caption</b>	<b>Page</b>
A V.1	Quantization Levels of Luminance Signal	A V-2
A V.2	Quantization Levels of Color Difference Signals	A V-2
A V.3	Display Compatibility Mode	A V-5

This page is intentionally left blank



A V.1 CCIR Level Assignment

---

**A V.1 CCIR Level Assignment**

The CCIR recommendation 601.1 defines certain reference binary levels for a uniformly quantized pcm image having 8 bits per sample. Luminance samples are represented by a positive binary number and colour difference samples by an offset binary number. The total nominal excursion of the luminance signal corresponds to 220 quantization levels, with black corresponding to level 16, and nominal white to level 235 (Figure A V.1).

There is an unequal quantization margin above and below the nominal signal because there is a greater variation in the nominal white level than in the nominal black level, and the effect of clipping the overshoot will be more perceptible in the white region.

As the luminance signal occupies 220 levels and as black is at level 16, the digital luminance signal  $Y_d$  is given by:

$$Y_d = 219 * Y + 16$$

where  $Y$  is the analogue luminance signal expressed as a fraction of unity, and  $Y_d$  is the corresponding level number after quantization to the nearest integer value (see Figure A V.1).

The colour-difference signals each occupy 225 levels in the central part of the quantization scale, with zero signal corresponding to level 128 (see Figure A V.2). The digital colour-difference signals  $U_d$  &  $V_d$  are therefore given by

$$\begin{aligned} V_d &= 224 * (0.713 * (R - Y)) + 128 \\ U_d &= 224 * (0.564 * (B - Y)) + 128 \end{aligned}$$

where  $V_d$   $U_d$  are quantized to the nearest integer value.

$R - Y$ ,  $B - Y$  are the colour-difference analogue values of any colour expressed as a fraction of unity.

A V.1 CCIR Level Assignment

---

Figure A V.1 **Quantization Levels of Luminance Signal**

LEVEL		BINARY	HEX
255	.....	(11111111)	FF
235	..... White .....	(11101011)	EB
16	..... Black .....	(00010000)	10
0	.....	(00000000)	0

Figure A V.2 **Quantization Levels of Color Difference Signals**

LEVEL		BINARY	HEX
255	.....	(11111111)	FF
240	..... Maximum .....	(11110000)	FO
128	..... Zero .....	(10000000)	80
16	..... Minimum .....	(00010000)	10
0	.....	(00000000)	0

## A V.2 Guidelines for 525/625 Image Interchange

### 2.1 Introduction

It is mandatory that all CD-I discs 'work' on all CD-I decoders. By this is meant that:

- all essential information is presented to the user.
- the user has access to all functions of the application on the disc.

This capability is the responsibility of the application, with some assistance from the decoder hardware and system software.

The implications of this for images are:

- All images are 'correctly' displayed on all decoders, whether 525 or 625 line format, that is they must not be 'scrambled'.
- The vital information in the safety area of all images must be fully visible on all decoders.
- It must be possible to position the cursor and pointing device correctly on all decoders.
- Additionally, Motion video must work correctly on all decoders, whatever their field rate.

This appendix gives guidelines on how the application can achieve these goals.

## 2.2 Image Interchange

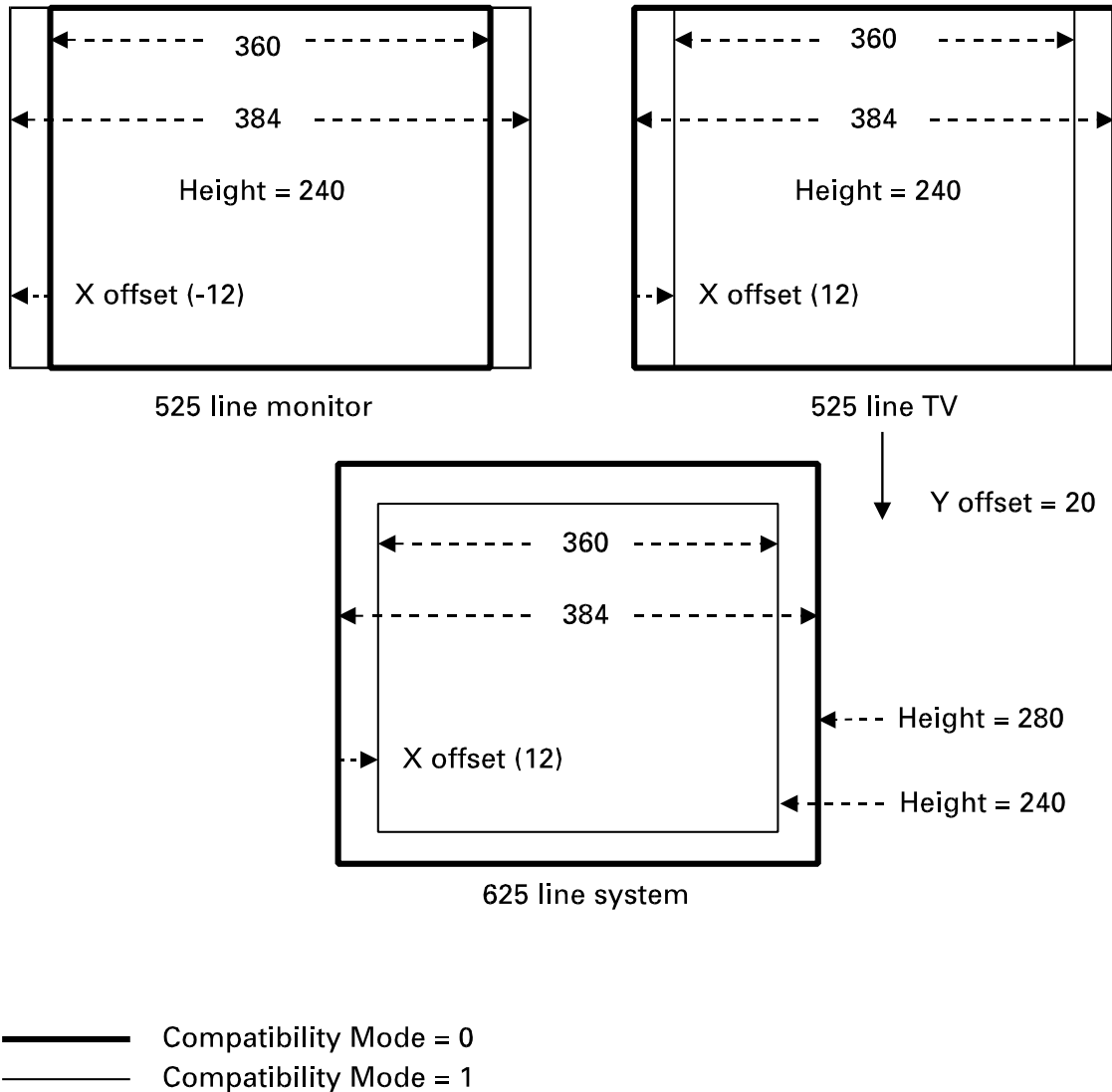
### 2.2.1 Display Compatibility Mode

The main function provided by the decoder to assist with image interchange is the ability to change the dimensions of the full-screen display according to the setting of a flag called 'Compatibility Mode'. This function is specified in V.4.8. The Compatibility Mode is set by the UCM function DC\_SetCmp.

The dimensions of the full-screen display in the two cases, and the offsets in pixels of the top left corner of the display for Mode = 1 from that for Mode = 0 are given in tabular form in V.4.8. Figure A V.3 below show the same information in diagrammatic form.

Note that the pixel clock frequency, and thus the pixel aspect ratio, remains the same when the Compatibility Mode is changed.

Figure A V.3 Display Compatibility Modes



The X offsets (i.e. -12, 12, and 12 respectively) and Y offsets shown are those recommended in order to ensure the visibility of the entire safety area.

**A V.2 Guidelines for 525/625 Image Interchange**

---

**2.2.2 Fitting the Image to the Display**

The image may have different width from that of the display, different height, or both. Of these, the most important is the width. The display width must be the same as that of the image, to support the simplest case where an image does not have a Display Line Start Pointer on each line. A difference of height simply means that less lines are displayed.

Accordingly, the application should set the Compatibility Mode such that the display width matches that of the image. If the image height is greater than the display, it should also arrange to move the image up, in order to centralize the safety area, by starting to display at a point 20 lines lower in the image.

The application can find out which display format is supported by the decoder by interrogating the Device Status Descriptor for the Display Monitor (see Appendix VII.2).

Details of what the application must do for specific cases are as follows:

**525 line images (360 x 240)****525 Monitor**

Set Compatibility Mode to 0

**525 TV**

Set Compatibility Mode to 1

**625 Display**

Set Compatibility Mode to 1

**625 line and Compatible Images (384 x 280)****525 Monitor**

Set Compatibility Mode to 1.

When the FCT is linked to the LCT, it must be linked to a line number 20 lines greater than the starting line for a 625 display.

This line of the LCT must contain a Display Line Start Pointer pointing to the start of its associated line of the image.

**525 TV**

Set Compatibility Mode to 0

When the FCT is linked to the LCT, it must be linked to a line number 20 lines greater than the starting line for a 625 display. This line of the LCT must contain a Display Line Start Pointer pointing to the start of its associated line of the image.

**625 Display**

Set Compatibility Mode to 0.

**525 line TV images (384 x 240)**

As well as the main specified image formats above, 525 line TV format images may be encoded and displayed, as follows:

**525 Monitor**

Set Compatibility Mode to 1

**525 TV**

Set Compatibility Mode to 0

**625 Display**

Set Compatibility Mode to 0.

Insert 'dummy' LCT sections of 20 lines each at top and bottom of the LCT for the image. These should have display line start pointers pointing to (nominally) blank image strips of 20 lines each.

**Images wider than Full-screen**

The above is concerned with full-screen images. For images of width greater than full-screen, which by definition have a Display Line Start Pointer associated with each line, the display width must be set to the width of the portion of the image that the application wishes to have displayed.

### 2.2.3 The LCT and Visual Effects

When a 625 line image, of a height of 280 lines, is displayed on a 525 line display, with a height of 240 lines, not only are 40 lines of the image not displayed, but also 40 lines of the original LCT are not executed.

For this, the application has the option of which 40 LCT lines to ignore. It is recommended that the application links the FCT to line 20 of the LCT. This will cause the top or the bottom 20 lines of the LCT to be ignored. In any case, the Display Line Start Pointer should reference line 20 of the image, so that the center 240 lines of the original image are still displayed.

The primary difference between the options of which lines to ignore is how the application will write into the LCT. All writes must be done relative to the new beginning line. Any instructions in the LCT which are before the linked line are not executed. In any but the recommended case, the application must remember that there is an offset between drawmap lines and LCT lines.

The non-execution of lines at the bottom of the LCT is not a problem, as DCP instructions on a given line affect only that line and later lines, and not previous lines in the image or lines on later fields (see V.4.5.1). However, the non-execution of the lines before the linked line must be taken into consideration by the application.

As an example, a rectangular matte starting on line 15 requires two 'load matte register' instructions on line 15, and no further instructions on later lines. When displayed on a 525 line decoder, these instructions must be inserted on line 20 by the application.

On the other hand, as another example, a circular matte, which requires 'load matte register' instructions on each line, requires no special attention by the application for image interchange.

Of course, any DCP instructions which are intended to affect the whole image should be placed in the FCT, and are then independent of the display height.



#### 2.2.4 **Cursor Alignment**

The default cursor origin is always at the top left corner of the full-screen image. This is true whether the Compatibility Mode is 0 or 1. There is therefore no offset of the cursor from the image, except that which is caused by the 20 line vertical offset at link time that the application must apply to 625 line images on 525 displays.

The application can compensate for this by offsetting the cursor origin vertically by 20 lines, using the UCM function GC\_Org.

### 2.2.5 Pointer Alignment

All pointing devices deliver coordinates whose default origin is at the top left corner of the full screen display for Compatibility Mode = 0.

In order for the application to know at which point in the image the pointing device is pointing, it must know the offset of the top left corner of the image from this point.

This offset is composed of two components.

- the offset of the top left corner of the full-screen display from this point. This is returned by a call to DC\_SetCmp: it is zero for Compatibility Mode = 0.
- The 20 line vertical offset of the image from the full-screen display applied by the application when displaying a 625 line image on a 525 line decoder.

By combining these two offsets, the application can know the offset to apply to the pointer. The origin of the pointer can be offset accordingly by a call to the UCM function PT\_Org.

### 2.2.6 More than One Standard on a Disc

The discussion of image interchange above assumes that images have been prepared for one particular standard, and describes how to display them on all decoders.

An alternative is for the application to have images of more than one standard on disc, and to choose them according to the decoder it finds itself on.

An extension of this is the fully adaptive application, which positions and sizes what it displays to suit the dimensions of the display available.

Applications of this kind may make use of the full 250 line safety area of 625 line displays.

### 2.3 Field Timing

An intrinsic difference between 525 line/60 Hz and 625 line/50 Hz displays is that they have different field periods (16.7 and 20 ms respectively). This gives rise to two potential 525/625 line interchange problems, i.e.

- what happens to motion video for one standard when it is displayed on the other? Examples of motion video are a rapid sequence of partial updates, or full-screen runlength animation.
- if part of the screen is updated every field, say to move an 'object', the speed of motion will be greater on 525 line systems.

There is nothing the decoder can do to help with these problems. It is entirely the responsibility of the author to ensure that he writes his application to be independent of the field period. Some techniques for doing this are as follows.

### 2.3.1 Motion video

The image update rate can never be precisely locked to even a single field rate, as the disc rotation and the display clock are not locked together. The retrieval of images from the disc and their displays are of necessity asynchronous.

The display of each successive image may be initiated by completion of loading the image from disc, if the images are spaced at regular intervals on the disc. If not, e.g. for runlength images, another scheme, such as regularly spaced trigger bits on the disc, must be used.

For any moving video a double or triple buffering scheme is necessary to prevent half-formed images being displayed. For partial screen moving video, the width of each image is generally less than that of the screen, so a partial update or similar copy operation must be performed between the disc buffer and the display memory.

If this copy operation takes less than one field period (i.e. 16.7 ms, as we must take the worst case), a single display memory may be used, with the use of the 'signal on display line' DCP instruction to synchronize the copy to the field scan and thus prevent flicker. If not, two or more display memories are required, displayed alternately by changing display pointers in the active DCP.

In both cases, each successive image is synchronized to the first available field, whether at 50 or 60 Hz. For say 12.5 images/ sec, on 625 systems each image is displayed for 4 fields: on 525 systems the average is 4.8 fields, in a pattern 5,5,5,5,4. The visual results are generally acceptable.

For full-screen runlength animation, no copying is necessary, but the successive images are of different length in memory, and may take very different times to load from disc. A multiple buffer scheme may be used to even out arrival and display rates.

### 2.3.2 **Object motion**

Never lock anything to field rate. Always synchronize update and display operations to an independent reference, such as the system clock, and update the display on the next field, as indicated by a 'signal on display line' DCP instruction, to avoid flicker.

**CD-RTOS<sup>1</sup> Technical Manual**

This appendix refers to the OS-9<sup>2</sup>/68000<sup>3</sup> Technical Manual<sup>4</sup>. Only certain parts of the OS-9/68000 Technical Manual are pertinent to the CD-RTOS Base Case System. The chapters of the OS-9/68000 Technical Manual which correspond to the Base Case version of CD-RTOS are:

- System Overview
- The Kernel (except Sysgo and Extension Modules)
- OS-9 Input/Output System
- Interprocess Communications
- Trap Handlers
- The Math Module
- Appendix A (except Sysgo and RBF/SCF/SBF device descriptor)
- Appendix B (only Pipeman path descriptor)
- Error Codes
- OS-9 System Calls

All of these chapters of the OS-9/68000 Technical Manual are completely a part of the CD-RTOS Base Case System with the exception of all but one of the file managers referred to in the OS-9 Input/Output System chapter. The CD-RTOS Base Case System only includes the pipe file manager, i.e. Pipeman.

---

<sup>1</sup> CD-RTOS is the copyright protected property of Philips Consumer Electronics B.V., and Sony Corporation.

<sup>2</sup> OS-9 © is a trademark of Microware Systems Corporation.

<sup>3</sup> 68000 ©, 68010 ©, 68070 ©, is a trademark of Motorola Inc.

<sup>4</sup> This manual is revision J and reflects version 2.4 of the OS-9/68000 Operating System. The manual, which is published by Microware Systems Corporation, is made available as a part of the CD-I Full Functional Specification.

This page is intentionally left blank



A VII.2 Table of Contents

---

**Appendix VII.2 CD-I Peripherals**

	<b>Page</b>
2.1 General	A VII.2-1
2.1.1 Scope	A VII.2-2
2.1.2 Introduction	A VII.2-2
2.2 CD-IX	A VII.2-4
2.2.1 Objective	A VII.2-4
2.2.1.1 CD-IX Configuration	A VII.2-4
2.2.1.2 CD-IX Extension Options	A VII.2-5
2.2.1.3 CD-IX Future Options	A VII.2-6
2.2.2 CD-IX Configuration	A VII.2-6
2.2.2.1 CD-I Base Case	A VII.2-6
2.2.2.2 Display System	A VII.2-6
2.2.2.3 Keyboard	A VII.2-7
2.2.2.4 Floppy Disc Drive (3.5 inch)	A VII.2-7
2.2.3 CD-IX Extension Options	A VII.2-8
2.2.3.1 Printer	A VII.2-8
2.2.3.2 Modem	A VII.2-8
2.2.3.3 Hard Disk Drive	A VII.2-9
2.2.3.4 RAM Disk	A VII.2-9
2.2.3.5 MIDI Interface	A VII.2-9
2.2.3.6 Graphics Co-processor	A VII.2-10
2.2.3.7 Arithmetic Co-processor	A VII.2-10
2.2.3.8 Local Area Network	A VII.2-10
2.3 Implementation Aspects of CD-I Peripherals	A VII.2-11
2.3.1 General	A VII.2-11
2.3.2 'Built-in' Peripherals	A VII.2-11
2.3.3 System Bus Based Peripherals	A VII.2-12
2.3.4 Interface Based Peripherals	A VII.2-12

This page is intentionally left blank

## A VII.2 Table of Contents

	<b>Page</b>
2.4 Hardware Configuration Status Descriptor	A VII.2-13
2.4.1 General	A VII.2-13
2.4.2 Device Status Descriptors	A VII.2-13
2.4.2.1 Device Type Field	A VII.2-14
2.4.2.2 Device Name Field	A VII.2-14
2.4.2.3 Device Parameter Field	A VII.2-14
2.4.2.4 Multiple Presence	A VII.2-15
2.4.3 CSD Access	A VII.2-16
2.4.4 Adding New Devices to the CSD	A VII.2-16
2.5 Functional Specification of CD-I Peripherals	A VII.2-17
2.5.1 Pointing Devices	A VII.2-17
2.5.1.1 Classes of Pointing Device	A VII.2-17
2.5.1.2 Pointer Coordinates	A VII.2-18
2.5.1.3 Pointer Buttons	A VII.2-18
2.5.1.4 Pointer Input Functions	A VII.2-19
2.5.2 Keyboards for Latin Alphabets	A VII.2-20
2.5.2.1 Keygroups	A VII.2-21
2.5.2.2 Classes of Keyboard	A VII.2-23
2.5.2.3 National Versions	A VII.2-24
2.5.2.4 Code Assignments	A VII.2-25
2.5.2.5 Keyboard Driver	A VII.2-26
2.5.2.6 Keyboard Input Functions	A VII.2-27
2.5.3 Player Control Keys	A VII.2-28
2.5.3.1 Keygroup	A VII.2-28
2.5.3.2 Key Functions	A VII.2-28
2.5.3.3 Code Assignments	A VII.2-30
2.5.3.4 Player Control Key Driver	A VII.2-30
2.5.3.5 Player Control Key Input Functions	A VII.2-31

This page is intentionally left blank

A VII.2 Table of Contents

---

	<b>Page</b>
2.6 Device Status Descriptors	A VII.2-32
2.6.1 General	A VII.2-32
2.6.2 Base Case Devices	A VII.2-32
2.6.2.1 System	A VII.2-32
2.6.2.2 CD-Control Unit	A VII.2-32
2.6.2.3 Audio Processor	A VII.2-33
2.6.2.4 Video Output Processor	A VII.2-33
2.6.2.5 Non-volatile Random Access Memory (NVRAM)	A VII.2-34
2.6.2.6 Pointing Device	A VII.2-34
2.6.3 Base Case Peripherals	A VII.2-35
2.6.3.1 CD-Player	A VII.2-35
2.6.3.2 Audio Set	A VII.2-35
2.6.3.3 Display Monitor	A VII.2-36
2.6.4 Base Case Device "Pipe"	A VII.2-37
2.6.4.1 Pipe Device	A VII.2-37
2.6.5 Peripheral Extensions	A VII.2-38
2.6.5.10 Keyboard	A VII.2-39
2.6.5.20 SCF Devices	A VII.2-40
2.6.5.30 RAM device	A VII.2-41
2.6.5.40 RBF Floppy Disk	A VII.2-41
2.6.5.50 RBF Hard Disk	A VII.2-41
2.6.5.60 PCFM Floppy Disk	A VII.2-41
2.6.5.70 PCFM Hard Disk	A VII.2-42
2.6.5.80 SBF Tape Device	A VII.2-42
2.6.5.90 MPEG Video Device	A VII.2-42
2.6.5.91 MPEG Audio Device	A VII.2-43
2.6.5.100 NFM LAN Device	A VII.2-43

This page is intentionally left blank

List of Illustrations

---

<b>Fig. No.</b>	<b>Caption</b>	<b>Page</b>
A VII.2.1	CD-I Configuration	A VII.2-2
A VII.2.2	CD-IX Extension Options	A VII.2-5
A VII.2.3	Minimum Keyboard Configuration	A VII.2-23
A VII.2.4	Key Code Assignments	A VII.2-25

This page is intentionally left blank



## A VII.2 CD-I Peripherals

---

### A VII.2 CD-I Peripherals

#### 2.1 General

##### 2.1.1 Scope

This appendix on CD-I peripherals gives a functional specification of the input and output devices of CD-I systems.

Sections A VII.2.1 - 2.4 deal with a CD-I system configuration beyond the Base Case called CD-IX (A VII.2.2), the general aspects of CD-I peripherals (A VII.2.3) and gives details on the use of the Configuration Status Descriptor (CSD, A VII.2.4).

The CSD has a major function in CD-I systems as it allows applications to ascertain the actual hardware configuration.

Section A VII.2.5 gives the functional specification of particular CD-I peripherals and section A VII.2.6 lists the code assignments, and the particular meanings of device dependent parameters, for each of the functional devices. These two sections will be added to as new devices are defined.

The major goal of this functional specification is to provide a consistent approach for extended CD-I configurations. In this respect, the Configuration Status Descriptor (CSD) is an essential tool for application designers.

## A VII.2 CD-I Peripherals

---

### 2.1.2 Introduction

A CD-I Base Case can be, for purposes of discussion, broken down (see Figure A VII.2.1) into a CD-I Base Case decoder and CD-I Base Case peripherals. The CD-I Base Case decoder is the nucleus of any CD-I system. It comprises the microprocessor, the audio processor, video processor, memory and the CD-I control unit. CD-I peripherals are input and output devices that can be used in connection with the CD-I Base Case Decoder.

Following the above interpretation, the Base Case CD-I system is configured as a CD-I Base Case decoder with CD-I Base Case peripherals such as a monitor and a pointing device. The Base Case CD-I configuration and the use of the Configuration Status Descriptor (CSD) are **mandatory**.

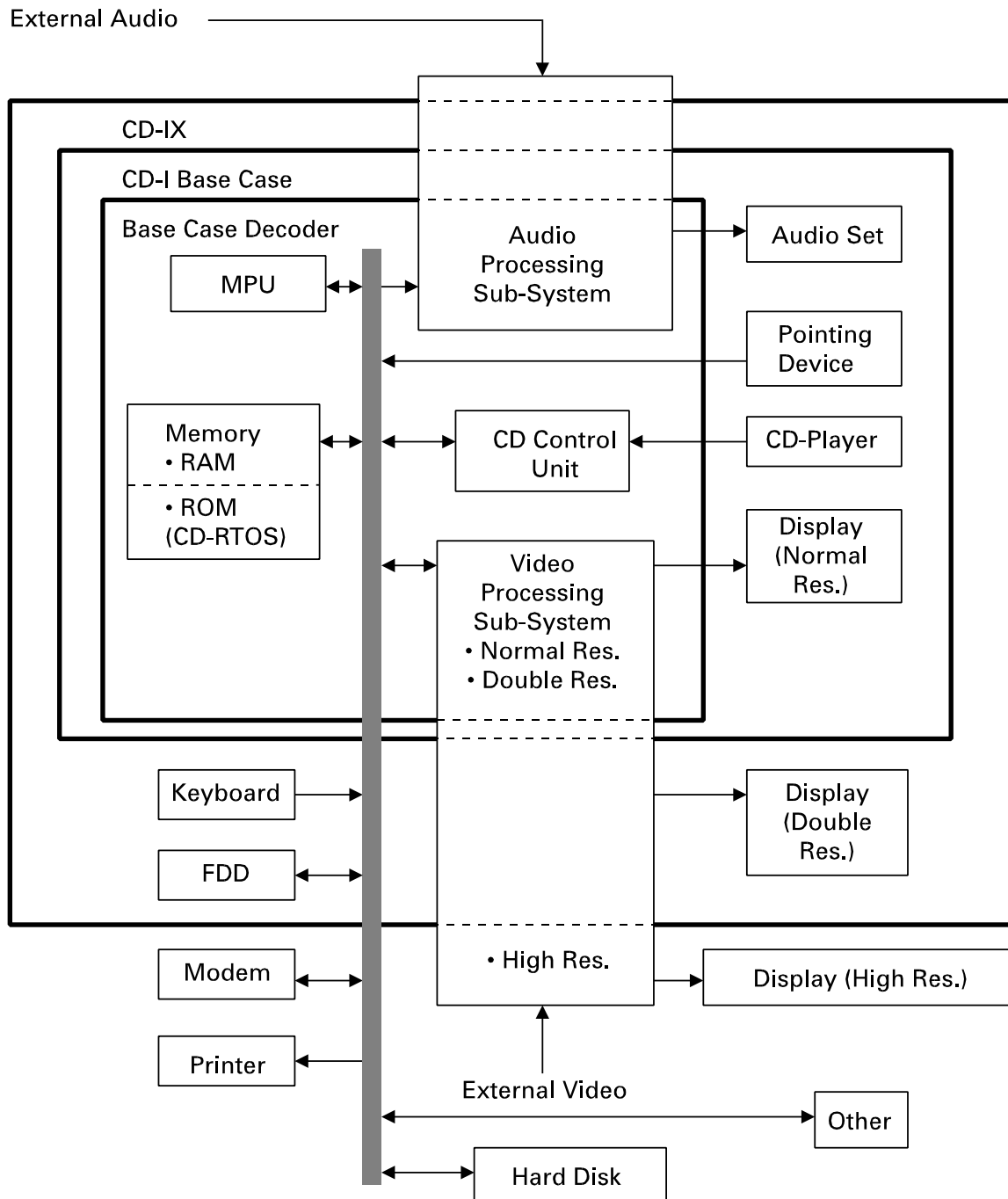
A basic principle for extendability as given here is that any CD-I peripheral is considered independent of the availability of any other peripheral device. As such, all peripherals are specified as self-contained. Clearly, this approach will lead to a high number of configurations. For this reason, preferred configurations can be proposed for certain application areas.

One such configuration for a personal information system is designated CD-IX and specified in A VII.2.2. CD-IX will always include the functionality of the CD-I Base Case. This proposed configuration is fully compatible, independent of whether it is available as an integrated system or if it is assembled from the individual peripherals.

The relationship between the different CD-I configurations is indicated in Figure A VII.2.1 below.

A VII.2 CD-I Peripherals

Figure A VII.2.1 CD-I Configuration



## A VII.2 CD-I Peripherals

---

### 2.2 CD-IX

This section describes the CD-IX configuration and the devices which this includes. Functional specifications for CD-IX and extension devices are given in A VII.2.5 and will be added to as and when required.

#### 2.2.1 Objective

A Base Case CD-I system can be expanded to include additional input/output devices and hardware options. The result can be a package which may be, for example, a personal information system, an electronic publishing system, or a car navigation system.

In order to make application software development efficient and non-hardware specific, developers of software for extended CD-I systems need to know the minimum level of functionality that can be expected. They also require standardized interface definitions (i.e. virtualization) of the various options which may exist.

The purpose of this CD-IX specification is to ensure an orderly and compatible development of extended CD-I systems. Systems meeting the requirements of this specification are designated 'CD-IX'.

##### 2.2.1.1 CD-IX Configuration

The basic configuration for the CD-IX system is given below. All devices need be present for a system to be used as a CD-IX system. The manufacturer must provide the necessary hardware decoding capability and interfaces as well as software managers and drivers in accordance with A VII.2.3 and 2.4.

#### CD-IX Configuration

- The CD-I Base Case Configuration.
- Double Resolution display.
- Full alphanumeric keyboard.
- One 3.5 inch Floppy Disk Drive.

## A VII.2 CD-I Peripherals

## 2.2.1.2 CD-IX Extension Options

The following options are those which can be immediately supported by the system software of CD-IX, and, therefore, may be provided as a hardware manufacturer's option. Any combination of options may be used. Each option can be built-in to CD-IX or installed later as a plug-in option.

All necessary software modules (e.g. device drivers, etc.), needed to support any device must be provided with the necessary hardware interfaces.

**CD-IX Extension Options**

- RAM extension module
- Hard disk drive
- Printer
- Modem
- MIDI interface
- Co-processors
- Local Area Network

An overview of the above mentioned configurations is given in Figure A VII.2.2.

Figure A VII.2.2 CD-I Extension Options

Device	CD-I Base Case	CD-IX	CD-IX Extentions
RAM size	1 MB	1 MB	further expansions
Clock Calendar	yes	yes	yes
Co-processors	no	no	yes
Display Type	Normal	Double	High
Pointing device	yes	yes	yes
Keyboard	Application <sup>1</sup>	yes	yes
3.5 inch FDD	no	yes	yes
Printer	no	no	yes
Modem	no	no	yes
Hard disk	no	no	yes
MIDI	no	no	yes
LAN interface	no	no	yes

<sup>1</sup> If no keyboard is available the application is responsible for obtaining the necessary input by using the display and the X-Y pointing device.

## A VII.2 CD-I Peripherals

---

### 2.2.1.3 CD-IX Future Options

CD-IX has, in accordance with the extension philosophy of CD-I, the possibility of further extendability covering a broad range of present and future devices. This document describes devices which can, in principle, be supported now as well as how additional devices may be supported in the future. Compatibility is facilitated by the inherent modularity of CD-RTOS and the use of the Configuration Status Descriptor (CSD) mechanism.

### 2.2.2 CD-IX Configuration

#### 2.2.2.1 CD-I Base Case

CD-IX includes the CD-I Base Case system's functionality. The following Base Case devices are supported by the CD-RTOS file managers (see VII.2)

- CD-Control Unit
- Audio Processor
- Video Processor
- Non-volatile RAM
- Pointing Device
- Keyboard (optional)

It is up to each manufacturer to include additional file managers to CD-RTOS to support the additional CD-I peripherals. CD-RTOS compatible file managers for certain groups of devices are the OS-9 file managers for serial character files, random block files and serial block files (see A VII.1).

#### 2.2.2.2 Display System

The CD-I Base Case is capable of Normal and Double Resolution. However, the CD-I Base Case display is a Normal Resolution display. A Double Resolution display is required for CD-IX.

## A VII.2 CD-I Peripherals

---

### 2.2.2.3 Keyboard

A full alphanumeric keyboard is required for a CD-IX configuration. Limited functions can be obtained by the use of smaller keyboards and keypads. Different keyboards will be required for different alphabets and writing systems. In particular keyboards for Latin alphabets and Kanji will have very different capabilities.

### 2.2.2.4 Floppy Disk Drive (3.5 inch FDD)

Floppy Disk Drives (FDD) give CD-IX systems a means to permanently store a reasonable amount of user data.

A single medium and format standard is necessary in order to ensure disk interchangeability between different models and brands of CD-IX systems. At least one 3.5 inch FDD is required. The standard OS-9 disk format is specified (see A VII.1). However, the system shall also be capable of reading files, writing files, and accessing directories on standard MS-DOS format 3.5 inch Floppy Disks.

It should be noted that CD-I real-time files cannot be 'played' from magnetic media due to the different capabilities and characteristics of the two media.

### 2.2.3 CD-IX Extension Options

#### 2.2.3.1 Printer

CD-I will be a medium with very high non-textual visual information content. Therefore, printers which support only a character mode will not be able to provide a reasonable hard-copy of images presented on the video display. This also causes problems for application software developers, who effectively have to design two presentation modes, one for video displays and one for hard-copy.

On the other hand, printers with full image writing capabilities can provide (with the appropriate virtual interface) a direct image of the video display with mixed text and pictures. Printers with this capability range from low cost dot-matrix types up to high performance laser printers.

Image printers can use the various font styles, sizes, and national character sets inherently provided by the CD-I display system. Use of image printers is widely acknowledged as a requirement for electronic publishing, which is a potential CD-I application.

CD-IX Extension will only support image printers.

#### 2.2.3.2 Modem

Data communications capabilities provide important features, such as access to videotex on-line information services, home shopping and other similar capabilities.

This interface should provide a full RS232 serial, bidirectional interface with control signals suitable for control of auto-answer/auto-dial modem.

The baud rate must be user-selectable and capable of reversible, split-rate operation for videotex type systems.



## A VII.2 CD-I Peripherals

---

### 2.2.3.3 Hard Disk Drive

Various types and sizes of magnetic hard disks may be used with CD-IX. The following points apply to hard disks:

- capacity: no restriction
- standard OS-9 logical file format

The type of hardware interface used is up to the hardware manufacturer. The file manager and file format are specified in A.VII.1.

### 2.2.3.4 RAM Disk

A specialized driver, controlled by the random block file manager (see A.VII.1), allows RAM memory to be used as very high speed temporary file storage, which is typically used to hold small working files.

Memory allocated for RAM disk is not available for use by applications programs as general purpose working storage. However, the user can choose whether or not to use memory for this function.

### 2.2.3.5 MIDI Interface

MIDI (Musical Instrument Digital Interface) hardware may be provided as an option. MIDI allows computers to interface to synthesizers and other electronic music devices. The MIDI interface is considered as a de facto standard interface for musical instruments.

The available OS-9 MIDI driver module will permit real-time recording of instruments/devices, real-time playback of up to 16 instruments simultaneously, and block storage/retrieval of MIDI data to external devices. Specifications are based on the current International MIDI Association (IMA) documents.

## A VII.2 CD-I Peripherals

---

### 2.2.3.6 Graphics Co-processor

The use of a specialized graphics co-processor can provide a considerable increase in system speed and performance with only modest additional cost. Experience has shown the speed improvement to be in the order of five to ten times for the basic operations involved. The overall system improvement is application dependent.

Due to the modular architecture of CD-RTOS, this options can be used to replace software routines while applications programs remain compatible and automatically benefit from the performance gain.

A graphics co-processor provides high-speed drawing primitive functions which substitute for some or all the software routines included in the UCM and video display driver modules (see VII.2.3). The functions of a particular co-processor must match CD-I screen resolution and UCM primitive functions.

### 2.2.3.7 Arithmetic Co-processor

An arithmetic co-processor provides high speed floating point arithmetic operations. A CD-RTOS 'Math' module which implements these routines can be replaced with an equivalent module which utilizes a co-processor. Any co-processor which supports the single and double precision formats given in the IEEE Standard for Binary Floating Point Processors (Task Group P754) may be used.

### 2.2.3.8 Local Area Network

Local Area Network (LAN) is a capability which must be available to penetrate educational and certain semi-professional and office automation markets. A CD-RTOS compatible LAN subsystem called Network File Manager (see A VII.1) is available.

NFM combines the individual file systems of each processor into a single logical file system. This permits any system on the network to access files and devices on other systems as if they were local. The network system is completely transparent to CD-I applications software. A security mechanism is provided.

## 2.3 Implementation Aspects of CD-I Peripherals

### 2.3.1 General

Essential for each CD-I peripheral is that it has to make available to the system its own Device Status Descriptor (DSD), device driver and file manager.

There are several methods for implementing an extended CD-I system. The three methods described here are not exclusive to CD-I, but are well-known from other computer configurations. The methods can be used in any combination. Only these three are mentioned here because these are being considered as standard methods supported by CD-RTOS modules for CSD compilation.

The standard methods are:

- (1) 'Built-in' peripherals;
- (2) System bus based peripherals; and
- (3) Interface based peripherals.

Although it is technically feasible to restrict oneself to one of these methods, it is likely that a CD-I system may deal with a mixture of methods.

### 2.3.2 'Built-in' Peripherals

From a system point of view, this is the easiest method to implement.

The additional system software modules for these peripherals and information from which the required DSDs are compiled have to reside in system ROM.

When a CD-I peripheral is built-in, it is completely up to the hardware manufacturer how to integrate the CD-I peripheral into his design, as long as care is taken with respect to the requirements of the CSD. As such this system is essentially a closed system and is acquired as a ready-to-use unit.

---

## A VII.2 CD-I Peripherals

---

### 2.3.3 System Bus Based Peripherals

System bus based peripherals are CD-I peripherals that are designed to be plugged into extension slots of the system bus. A characteristic of this type of extension is that the CD-I Base Case has direct control over the CD-I peripherals.

The additional system software modules for operation of the peripherals and information from which the DSDs are compiled have to reside in separate ROM associated with the system bus based peripheral.

Mechanical and electrical specification of a system bus slot interface is up to each hardware manufacturer.

### 2.3.4 Interface Based Peripherals

Interface based peripherals refer to peripherals that are connected to the CD-I system through the use of (de facto) standard or proprietary interfaces. A major difference between interface based peripherals and the 'built-in' or system bus based peripherals is the non-direct control of the peripherals.

These interfaces include the RS232 interface, the Centronics interface and the SASI/SCSI interface.

Compared to the preceding methods, this method is a rather complicated one to handle. This is so because the interface is built-in and the peripheral is external without its own ROM. In general, such interfaces are well defined in their mechanical design and electrical characteristics. Although these interfaces are mostly designed for a well-defined objective, in practice they are often used for other purposes: e.g. an RS232 modem communication interface is often used as a printer interface or as a port to a data processing system.

These interfaces require that the supporting software modules, including information from which the DSDs are compiled, reside in the system ROM as for the built-in peripherals. However, this will lead to restrictions on compatibility of certain functional extensions of different brands and different models.

## 2.4 Hardware Configuration Status Descriptor

### 2.4.1 General

It is important for an application program to be able to determine the configuration of a CD-I system. To facilitate this CD-RTOS provides the Configuration Status Descriptor (see VII.1.3) which provides information about each device on the system whether it is a Base Case device or an extension beyond the Base Case. Each device is represented in the CSD by its Device Status Descriptor (DSD).

The CSD is stored in non-volatile RAM (see VII.1.3.2) in a file named "csd". Application programs may access the CSD information by opening and reading this file.

The formats for each field in the DSD are further specified in A VII.2.4.2 and completed with the actual coding for typical devices in A VII.2.6.

### 2.4.2 Device Status Descriptors

Each Device Status Descriptor (DSD) corresponds to one device and comprises three fields.

- Device Type** : This field contains an integer representing the general class of functional devices to which this device belongs. Examples might be: optical file storage devices, hardcopy printer, communications device, magnetic disk, pointing device, keyboard, sensor, etc.
- Device Name** : This field contains the name used to access the device.
- Device Parameters** : This field is used to provide complete information to identify the functional performance of the device. The contents are specific to a particular device type.

Each DSD comprises bytes with values in the range \$20 to \$7E and is terminated by a carriage return (\$0D) character.

## A VII.2 CD-I Peripherals

---

### 2.4.2.1 Device Type Field

The device type field is a string of numeric characters (0-9) terminated by a colon (":") to identify the function of the device.

The device codes are allocated according to the following rules:

<b>Device Type Code</b>	<b>Meaning</b>
0 to 9	Base Case devices
10 and above	Extension devices

### 2.4.2.2 Device Name Field

The device name field is a string of characters terminated by a colon (":").

For most devices the device name will be used by CD-RTOS to access the relevant CD-I device. This will be the actual name of the device descriptor and will begin with the "/" character. This allows an application to determine the name of a device from the device type. For others, not controlled by CD-RTOS, the name only identifies the DSD to the application. These device names are unique, even if more than one type of a particular device is connected.

The only Device Name which is defined for the CD-I Base Case system in this specification is "/nvr" for the NVRAM (see VII.2.4.2).

### 2.4.2.3 Device Parameter Field

The device parameter field contains an open-ended list of parameters to identify and/or to set the functional behaviour of the device identified by its device type. These parameters are interpreted by an application or driver and are unique to each device. The format is designed such that it can handle a multitude of parameters and does not restrict future enhancements. The parameter definitions are in readable format, using the ISO 8859-1 character set (codes \$20 to \$7E)

## A VII.2 CD-I Peripherals

---

Each entry in the Device Parameter Field has one of the following three formats:

- 1) Two character Parameter Identifier used as a boolean flag to indicate one of two particular capability options.

For example a monochrome display device would be defined as such by the parameter field "MO".

- 2) Two character Parameter Identifier followed by the "#" (\$23) character and a numeric parameter value. The value would comprise a variable length string of characters in the range \$30 to \$39.

For example, for a video processor capable of displaying 2 image planes, the relevant parameter field would be "PL#2".

- 3) Two character Parameter Identifier followed by the "=" (\$3D) character and an alphanumeric string.

For example, "KG="ALL"" in the Keyboard DSD would indicate that all Key groups were present.

Each parameter entry is terminated by a colon (":", or \$3A).

Where, for any parameter, two or more values can be present these can be separated by a comma (",", or \$2C).

The Parameter Identifier for all three formats must comprise two upper case alphabetic characters (\$41 to \$5A).

### 2.4.2.4 Multiple Presence

It is possible for more than one device of the same type to be present within the same hardware configuration. These will be individually identified by unique device names.

## A VII.2 CD-I Peripherals

---

### 2.4.3 CSD Access

The CSD is maintained in a file in NVRAM called "csd". It is made up of the set of DSDs merged together. Each DSD is terminated with a carriage return (\$0D). To access an individual DSD, it is **recommended** to open the file and parse through the entries using I\$ReadLn. (I\$ReadLn will read up to the next carriage return). When the appropriate DSD entry is found, the file should be closed.

To alter a parameter in a DSD, it is **recommended** to read the entire file into memory, make the appropriate changes and then to rewrite the entire file, setting the new file size.

### 2.4.4 Adding New Devices to the CSD

There are several ways that new devices may be added to the CSD. The appropriate manner depends upon the features of the device. If the device is added as a bus peripheral, the DSD for that device could be included in ROM on the board. When CD-RTOS searches ROM for modules at startup, it would find this module and add it to the DSD entries in the "csd" file in NVRAM.

If the peripheral is added as an interface based peripheral, the manufacturer may supply a disc with the device which contains a program to configure the CSD for the device.



## 2.5 Functional Specification of CD-I Peripherals

### 2.5.1 Pointing Devices

The pointing device allows the user to control the position of the screen graphics cursor (V.5.12) via the application. It is the responsibility of the application to position the cursor using the coordinates provided by the pointing device. Every Base Case player must include at least one pointing device.

#### 2.5.1.1 Classes of Pointing Device

There are four classes of pointing device:

a. **Absolute screen pointing devices**

Pointing devices which are used directly on the screen include light pens and touch screens. The screen position given by these pointers is tied directly to the display screen.

b. **Absolute coordinate pointing devices**

Pointing devices that deliver absolute coordinate positions for the screen pointer include graphics tablets and absolute coordinate mice. The output of these devices are the absolute positions of the pen in relation to the grid and its resolution.

c. **Relative coordinate pointing devices**

Pointing devices that deliver relative coordinate positions for the screen pointer include mouse and trackball devices. The output of these devices provides incremental values for both coordinates, relative to the last readout.

d. **Manoeuvring pointing devices**

Pointing devices that deliver control information to manoeuvre the screen pointer to its desired position include joysticks. The output of these devices gives the direction into which the cursor has to be moved with a fixed, or optionally, a variable velocity. At least 8 different directions, equally spaced around the full 360 degrees, must be available.

The class of pointing device in use is defined by the DSD.

---

## A VII.2 CD-I Peripherals

---

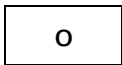
### 2.5.1.2 Pointer Coordinates

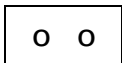
Each pointing device via the associated driver must output high resolution UCM coordinates to the application. The resolution of the device must be at least equivalent to normal resolution over the reduced screen area.

The pointer origin is, by default, at the top left corner of the full screen display. It is the responsibility of the application to set the origin for the coding format for which the disc is coded. This provides full compatibility when, for example, 525 line images are displayed using a 625 line format or vice versa.

### 2.5.1.3 Pointer Buttons

The pointing device will have two buttons or switches. The function of these is application dependent. The position and type of button or switch will vary from device to device. Where appropriate, it is **recommended** that the buttons be labelled as follows:

button 1 : 

button 2 : 

Furthermore it is recommended that where the two buttons have a left/right orientation, the left button is to be button 1. Where the buttons have a front/back or top/bottom orientation, the front or top buttons are recommended to be button 1.

For light pens and tablets where switches are built into the pen these switches will be identified as follows:

Switch 1 : in tip of pen

Switch 2 : on side of pen

The two switches 1 and 2 are equivalent to buttons 1 and 2 respectively.

#### 2.5.1.4 Pointer Input Functions

The following input functions are provided in UCM. For more details, see VII.2.3.5.

- PT\_Coord** returns pointer coordinates and pointer and button states
- PT\_SSig** defines the signal number to be returned when the pointer is moved or either button is depressed
- PT\_Relea** releases the pointer driver from the PT\_SSig function
- PT\_Pos** sets the position of pointing devices for classes c. and d. only. This has no effect for classes a. and b.
- PT\_Org** sets the pointer origin to a new position relative to the default position (see A VII.2.5.1.2).

A VII.2 CD-I Peripherals

---

**2.5.2 Keyboard for Latin Alphabets**

The keyboard provides the user an alternative means of controlling applications or of text and data entry. This specification is intended for keyboards for use in countries for which the CD-I default character set (which conforms to ISO 8859-1) is appropriate.

**2.5.2.1 Keygroups**





The keys on the keyboard are divided into the following groups:

**Programmable Function Keys**

This group comprises eight (8) keys, labelled F1, ..., F8, which must be interpreted by the application. These keys will generate 32 unique codes when used in conjunction with the special keys (see below).

**Cursor Control Keys**

The cursor is a screen pointer that indicates the "active" position on the screen for data input. The cursor control keys can, depending on the application, be regarded as a crude manoeuvring pointing device. Depressing these keys does not automatically cause the specified action, it is up to the application to interpret these codes and cause the proper action to occur.

	<b>Key Label (recommended)</b>
<b>Cursor Up</b> - move the cursor one unit* upwards	
<b>Cursor Down</b> - move the cursor one unit downwards	
<b>Cursor Left</b> - move the cursor one unit left	
<b>Cursor Right</b> - move the cursor one unit right	

\* The size of a unit is application dependent, but typically may represent one character position for text, or a pixel for graphics applications.

A VII.2 CD-I Peripherals

---

These keys may be used in conjunction with the special keys to produce different codes, which may be used by the application, for example, to move the cursor a greater distance than one unit.





### Alphanumeric Keys

This group comprises a minimum of 48 keys (including space bar), which together with the special keys produce all alphabetic, numeric, and punctuation characters and special symbols, as defined in ISO 8859-1.

All keys in this group must be labelled both with the characters produced in normal and shifted modes and, if not included already, all characters with codes in the range \$21 to \$7E in the ISO 8859-1 character set. Alphabetic keys need only be labelled with the upper case character.

### Special Keys

These keys do not generate any codes when depressed. Instead they modify the code produced by other keys, when used in conjunction with them. The state of these keys is returned by the driver when requested (using KB\_Stat). This group includes:

	Key Label
<p><b>Shift</b> - this key while depressed instructs the keyboard driver to generate code values according to the shift mode.</p>	
<p><b>Caps</b> - this optional key selects upper case (shift) mode for the alphabetic characters and, depending on the national version, also numeric characters. This effect should also extend to the supershift state. Pressing this key a second time selects lower case (normal) mode.</p>	
<p><b>Supershift</b> - this key while depressed instructs the keyboard driver to generate code values according to the supershift mode and may be used in conjunction with the shift key. This allows the selection of characters that are not accessible from the normal or shifted mode.</p>	
<p><b>Control</b> - this key when used in conjunction with other keys, will produce codes in the range \$00 to \$1F. When used also with supershift the codes \$80 to \$9F are generated.</p>	

A VII.2 CD-I Peripherals

---

These keys are used in conjunction with the other keys and each other to produce the remaining codes in the character set.

**Formatting Keys****Key Label**

**Return** - this key is used to mark the end of an input string. It may also be used as "enter" or "select".



**Tab** - this key is used to signal to the application to move the cursor to the next tabulator position.



or



**Delete** - this key is used to signal to the application to delete the character before the cursor position and backspace the cursor.



**Escape key** - this key generates the escape code.

**Labelled Function Keys**

The following keys are optional for all classes of keyboard:

**Help** - this key signals to the application that the user requires more information at the current point in the application to proceed.



**Menu** - this key signals to the application to pause itself and return to a main menu to allow the user to resume, pause, or quit the application.



**Home** - this key signals to the application to, for example, position the cursor at the top left of the screen.



Note that either the recommended icon should be used as key label or text as defined in the DSD.

**Numeric Keys**

This optional group comprises the numeric keys 0 to 9, which may be used to duplicate the same keys in the alphanumeric group.

A VII.2 CD-I Peripherals

---

**2.5.2.2 Classes of Keyboard**

Keyboards may have different formats from small keypads with limited keys to full alphanumeric keyboards. The following table lists the 7 key groups and defines a minimum configuration for a full alphanumeric keyboard needed to meet the CD-IX specification. The key groups used are defined in the DSD for each CD-I decoder.

Figure A VII.2.3 **Minimum Keyboard Configuration**

Key Groups	Number of Keys	Keyboard Configuration for CD-IX
1 Programmable function keys	8	M
2 Alphanumeric keys	48 min	M
3 Special keys	3 or 4	M
4 Cursor control keys	4	M
5 Formatting keys	4	M
6 Labelled function keys	1 to 3	O
7 Numeric keys	10	O

M : Mandatory

O : Optional

## A VII.2 CD-I Peripherals

---

### 2.5.2.3 National Versions

Different versions of the keyboard may be needed to suit particular national requirements. This may result in the following differences between keyboards:

- (1) Differences in the physical layout of alphanumeric keys.
- (2) Certain alphanumeric character codes in the range \$21 to \$7E will not be available except in Supershift mode and some codes in the range \$A0 to \$FF will be available in normal and shifted modes.

It is **recommended** that the alphanumeric keys should always be labelled with characters whose codes are in the range \$21 to \$7E even when these are generated in Supershift mode. This may require up to four character legends for some keys.



## A VII.2 CD-I Peripherals

## 2.5.2.4 Code Assignments

The code assignments for the keys defined in A VII.2.5.2.1 are given below. The code value returned to the application for each key is, in general, modified by combining it with one or more of the special keys.

Figure A VII.2.4 Key Code Assignments (hex codes)

Key Group	Mode			
	Normal	Shift	Supershift	Control
<b>Programmable Function Keys</b>				
F1	80	88	90	98
F2	81	89	91	99
F3	82	8A	92	9A
F4	83	8B	93	9B
F5	84	8C	94	9C
F6	85	8D	95	9D
F7	86	8E	96	9E
F8	87	8F	97	9F
<b>Control Keys</b>				
Left	08	11	15	02
Down	0A	12	16	03
Up	0B	13	17	04
Right	0C	14	18	05
<b>Formatting Keys</b>				
Return	0D	0D	0D	0D
Tab	09	19	19	19
Delete	7F	0F	0F	1F
Escape	1B	1B	1B	1B
<b>Function Keys</b>				
Help	1C	1C	1C	1C
Menu	1D	1D	1D	1D
Home	1E	1E	1E	1E

Other combinations of these keys with special keys (except caps) will return the normal value.

## A VII.2 CD-I Peripherals

---

The code assignments for the alphanumeric keys are as defined by the ISO 8859-1 code table. These keys used in conjunction with Shift and Supershift generate the codes \$20 to \$7E and \$A0 to \$FF. The combination of keys needed to produce certain codes may depend on the national version. In general, however, supershift will be needed to generate codes in the range \$A0 to \$FF.

Those alphanumeric keys, which in shifted mode generate codes in the range \$40 to \$5F, can be used with Control and without Shift to generate codes in the range \$00 to \$1F.

The same keys used with Control and Supershift generate codes in the range \$80 to \$9F.

Note that these rules apply whatever national version is used. For example, even if the code \$5E requires supershift to be depressed then code \$1E must be generated when used with Control and \$9E when used with Supershift and Control.

### 2.5.2.5 Keyboard Driver

The driver converts the direct keyboard output into the code values defined in A VII.2.5.2.4. In addition, it must meet the following requirements:

1. Key up/down events must be detected by the driver even when other keys are down at the same time. This includes determining the state of the special keys.
2. Auto repeat for all keys except special keys, labelled function keys, programmable function keys and the Escape key. The auto repeat latency and frequency can be defined by the application. The application may read the DSD for the user's preferred values or define them to suit the application.
3. All key up/down and auto repeat events must be queued by the driver. The queue buffer must hold at least 256 events. When I\$Read is used to read the input from the keyboard, auto repeat events are converted to keydown events (i.e. a string of characters) corresponding keyup events are ignored and removed from the queue.

For information on the UCM Driver interface see VII.3.2.3.

### 2.5.2.6 Keyboard Input Functions

The following keyboard input functions are provided by UCM. For a full specification see VII.2.3.6.

<b>I\$Read</b>	- Read a character string
<b>KB_Ready</b>	- Check for Data Ready
<b>KB_Read</b>	- Read Keyboard Event
<b>KB_SSig</b>	- Send Signal on Keyboard data ready
<b>KB_Rel</b>	- Release device
<b>KB_Repeat</b>	- Set Keyboard Latency and Repeat Times
<b>KB_Stat</b>	- Determine Status of Keyboard

---

## A VII.2 CD-I Peripherals

---

### 2.5.3 Player Control Keys

Player Control Keys (PCK's) are keys that have specific predefined functions that are common use in audio and video systems. They can be located on the player itself and/or as a peripheral. If a CD-I player supports Player Control Keys, this function can be detected by the application by reading the proper DSD in the CSD. If a CD-I player supports Player Control Keys, then a minimum set of PCK's must be available for the application: Play, Stop, Pause, Next and Previous.

#### 2.5.3.1 Keygroup

For the Player Control Keys the Keygroup number in the DSD is 8.

If one or more keygroups, other than 8, are only partially implemented in a CD-I playback system and if the keycodes for the implemented keys from these keygroups are in the range of \$01-\$7F, then these keys may be defined in keygroup 8 as available keys for the application.

#### 2.5.3.2 Key Functions

The Player Control Keys group includes the following key functions:

**Play** - Starts play or replay (if it was already playing) the selected track, music or video sequence from the beginning. If the Play key is pressed when in pause mode, it will have the same effect as pressing Pause again.

**Stop** - Stops the Play.

**Pause** - Interrupting a Play, pressing it again will continue the play starting from the position where it was interrupted.

**Next** - Steps to the next track, photo, music, option, etc. of the chosen play state or option submode.

**Previous** - Steps to the previous track, photo, music, option, etc. of the chosen play state or option submode.

**Search Forward** - Runs at high speed forward through the track, music or video sequences, thus allowing the user quick recognition of the position in the programme.

A VII.2 CD-I Peripherals

---

**Search Backward** - Runs at high speed backward through the track, music or video sequences, thus allowing the user quick recognition of the position in the programme.

**Repeat** - Start or repeat a programmed part of a disc.

**Next Disc** - Advances the next disc to the active position within a multi disc playback system.

**Previous Disc** - Advances the previous disc to the active position within a multi disc playback system.

**Still Forward** - Goes to still picture mode and if pushed again, increments the picture one by one without sound.

**Still Reverse** - Goes to still picture mode and if pushed again, decrements the picture one by one without sound.

**Slow Forward** - Plays a video sequence with a speed below normal and without sound.

**Slow Reverse** - Plays a video sequence in reverse direction with a speed below normal and without sound.

A VII.2 CD-I Peripherals

---

**2.5.3.3 Code Assignments**

The code assignments for the keys defined in A VII.2.5.3.2 are given below.

Figure A VII.2.5 **Key Code Assignments for Player Control Keys (hex codes)**

Key function	Key code	PCK configuration
Play	\$80	M
Stop	\$81	M
Pause	\$82	M
Next	\$83	M
Previous	\$84	M
Search Forward	\$85	O
Search Reverse	\$86	O
Repeat	\$87	O
Next Disc	\$88	O
Previous disc	\$89	O
Still Forward	\$8A	O
Still Reverse	\$8B	O
Slow Forward	\$8C	O
Slow Reverse	\$8D	O

M : Minimum set of Player Control Keys

O : Optional

**2.5.3.4 Player Control Key Driver**

The Player Control Key driver converts the direct keyboard output into the code values defined in A VII.2.5.3.3.

A VII.2 CD-I Peripherals

---

**2.5.3.5 Player Control Key Input Functions**

The following Player Control Key input functions are provided by UCM. For a full specification see VII.2.3.6 and VII.2.3.7.

<b>I\$Read</b>	- Read a character string
<b>KB_Ready</b>	- Check for Data Ready
<b>KB_Read</b>	- Read Keyboard Event
<b>KB_SSig</b>	- Send Signal on Keyboard data ready
<b>KB_Rel</b>	- Release device
<b>KB_Repeat</b>	- Set Keyboard Latency and Repeat Times
<b>KB_Stat</b>	- Determine Status of Keyboard
<b>KB_Avail</b>	- Key(s) available for the application
<b>KB_NrAvail</b>	- Number of keycodes available for the application

A VII.2 CD-I Peripherals

---

**2.6 Device Status Descriptors****2.6.1 General**

This section defines the DSD codes for Base Case and extension devices. It will be added to from time to time for new devices.

If any parameter is missing from the DSD the default value is assumed.

**2.6.2 Base Case Devices**

This section defines the Device Type codes and Device Parameters for the Base Case devices which are controlled by CD-RTOS. The Device Names are defined by CD-RTOS.

There are DSD entries for physical devices as well as for the peripheral unit that is connected to the I/O device, for example, the CD Control Unit and the CD Drive itself. In the case that a player contains multiple CD or Video devices, the devices and their corresponding peripherals are identified by the DV parameter in their DSDs. The DSDs for the second CDCU and CD Drive would both contain the entry DV#1.

**2.6.2.1 System**

<b>Device Type Code</b>	0	
<b>Parameter String</b>	<b>Description</b>	<b>Default</b>
RV = "X.Y"	CD-RTOS revision level e.g. 1.0	(No default)

**2.6.2.2 CD-Control Unit**

<b>Device Type Code</b>	1	
<b>Device Parameters</b>		
<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
SP#n	Sector data processing delay in ms = n	27
DV#n	Device number (if multiple devices of this type)	0



A VII.2 CD-I Peripherals

---

**2.6.2.3 Audio Processor****Device Type Code** 2**Device Parameters**

<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
AM	Audio Mixing Unit present	none present
AD#n	Audio Processing delay in ms = n	27

**2.6.2.4 Video Output Processor****Device Type Code** 3**Device Parameters**

<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
EV	Capable of synchronizing to external video (extended case)	Not capable
PL#n	Number of planes = n	2
CM	Continuous mosaic (Pixel repeat factor may be any value 2-255)	2,4,8,16 only
CS#N <sub>H</sub> ,N <sub>V</sub>	Cursor horizontal (N <sub>H</sub> ) and vertical (N <sub>V</sub> ) size in pixels	16, 16
HS	Capable of high scan	Normal scan
HI	High resolution display for all display modes	No High Resolution
LS	Line Sequential scan	Interlace
LI=String	Number of lines given by string string = "525" : 525 lines string = "625" : 625 lines	(No default)
TV	Television display	Monitor
DV#n	Device number (if multiple devices of this type)	0

A VII.2 CD-I Peripherals

---

**2.6.2.5 Non-volatile Random Access memory (NVRAM)****Device Type Code** 4**Device Parameters**

<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
SZ#n	Size in kbytes given by n	8 kbytes

**2.6.2.6 Pointing Device****Device Type Code** 5**Device Parameters**

<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
CL=string	Class of pointing device given by string: "a": Absolute screen pointer "b": Absolute coordinate pointer "c": Relative coordinate pointer "d": Manoeuvring pointer	(No default)
HI	Capable of high resolution Resolution	Normal

A VII.2 CD-I Peripherals

---

**2.6.3 Base Case Peripherals**

This section defines the DSD Device Type codes and Device Parameters for Base Case devices which are not controlled by CD-RTOS

**2.6.3.1 CD-Player**

**Device Type Code**        6

**Device Parameters**

<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
SK#n	Max. seek time = n sec	3 sec
ND#n	n = number of discs for multi-disc changer	1 disc
DV#n	Device number (if multiple devices of this type)	0

**2.6.3.2 Audio Set**

**Device Type Code**        7

**Device Parameters**

<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
-------------------------	--------------------	----------------

A VII.2 CD-I Peripherals

---

## 2.6.3.3 Display Monitor

**Device Type Code** 8**Device Parameters**

<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
MO	Monochrome display	Color
RS=string	Resolution defined by string "D" = Double "H" = High	Normal
LI=string	Number of lines defined by string String = "525" : 525 lines String = "625" : 625 lines String = "525,625" : 525 or 625 lines for a multi-standard display monitor	Number of lines of corresponding Video Output Processor
HS	High scan	Normal scan
LS	Line sequential	Interlace
DV#n	Device number (if multiple devices of this type)	0
AR=string	Display Aspect Ratio. String = "4:3": The connected display is set to a 4:3 ratio.  String = "16:9": The connected display is set to a 16:9 ratio.	"4:3"

A VII.2 CD-I Peripherals

---

**2.6.4 Base Case Device "Pipe"**

This section defines the Device Type Code for the Base Case device "Pipe" which is controlled by CD-RTOS. The Device Name is defined by CD-RTOS

**2.6.4.1 Pipe device**

**Device Type Code**            9

<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
-------------------------	--------------------	----------------

A VII.2 CD-I Peripherals

---

**2.6.5 Peripheral Extensions**

This section defines the DSDs for peripheral extensions. New devices/peripherals will be added from time to time.

The devices described by device codes 20, 40, 50, 60, 70, 80 and 100 are "generic" devices for each of the OS9 file managers. The system calls to access these devices are described in A VII.1. Space is reserved for specific purpose devices of each class of file manager.

**Note:** Numbering of further sections allow for future peripheral extensions.

A VII.2 CD-I Peripherals

---

## 2.6.5.10 Keyboard

**Device Type Code** 10**Device Parameters**

<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
KG=string	Key groups defined by string: "1" : Programmable function keys "2" : Alphanumeric keys "3" : Special keys "4" : Cursor control keys "5" : Formatting keys "6" : Labelled function keys "7" : Numeric keys "ALL" : All keygroups	(No default)
AN=LA1 String	Alphanumeric keygroup according to ISO 8859/1	"LA1"
HE=String	Help key present; label defined by string	No Help key
HE	Help key present; label is icon	No Help key
ME=String	Menu key present, label defined by string	No Menu key
HO=String	Home key present; label defined by string	No Home key
RE=String	Return key label identified by string	Icon or blank
LA#n	Auto Repeat latency in ms given by n	(No default)
RP#n	Auto Repeat interval in ms given by n	(No default)

A VII.2 CD-I Peripherals

---

**2.6.5.11 Player Control Keys****Device Type Code** 11**Device Parameters**

<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
KG=string	keygroups defined by string: "1" : Programmable function keys "2" : Alphanumeric keys "3" : Special keys "4" : Cursor control keys "5" : Formatting keys "6" : Labelled function keys "7" : Numeric keys "8" : Player Control Keys "ALL" : All key groups	none

All keygroups, except 8, conform to the keyboard specification in Appendix VII.

**2.6.5.20 SCF Devices**

For one SCF port only one DSD with the Device Type Code 20 can exist.

**Device Type Code** 20**Device Parameters**

<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
CL=string	Class of SCF device given by string: "MDM": Modem	General SCF device



A VII.2 CD-I Peripherals

---

**2.6.5.30 RAM device**

Since a RAM device is not controlled by CD-RTOS, no '/' is added to the Device Name in the actual device name. "RAMxx" is used as Device Name for a RAM Device in the DSD, where xx is a 2-digit numerical value. The first DSD of type 30 will have "RAM00" as Device Name, the next DSD will have "RAM01" as Device Name, etc.

**Device Type Code**            30

**Device Parameters**

<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
SZ#n	Memory size, expressed as a multiple of 1024 x 1024 bits	4
CO#n	Memory color, where n indicates the color of the memory	1
AT#n	The typical access time of the RAM device. The time is expressed in ns The access time is the time from the assertion of ASn to the negation of the corresponding DTACKn signal.	100

**2.6.5.40 RBF Floppy Disk**

**Device Type Code**            40

**2.6.5.50 RBF Hard Disk**

**Device Type Code**            50

**2.6.5.60 PCFM Floppy Disk**

**Device Type Code**            60

A VII.2 CD-I Peripherals

---

**2.6.5.70 PCFM Hard Disk****Device Type Code** 70**2.6.5.80 SBF Tape Device****Device Type Code** 80**2.6.5.90 MPEG Video Device****Device Type Code** 90**Device Parameters**

<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
SZ#n	MPEG Video Buffer Size in Mbits	4
SD#n	$\tau(\text{mpeg,constant})$ in units of 10 msec. Range 0..7	0
DV#n	Device number (if multiple devices of that type)	0
SP#n	Number of Still Picture pages (see Chapter IX.6.6.4)	1
RV=string	FMV audio and video revision level. String="CTP1" : cartridge P1 String="CTP2" : cartridge P2	"CTP1"

A VII.2 CD-I Peripherals

---

**2.6.5.91 MPEG Audio Device****Device Type Code** 91**Device Parameters**

<b>Parameter string</b>	<b>Description</b>	<b>Default</b>
SD#n	$\tau$ (mpeg,constant) in units of 10 msec. Range 0..7	0
DV#n	Device number (if multiple devices of that type)	0

**2.6.5.100 NFM LAN Device****Device Type Code** 100

This page is intentionally left blank

	<b>Page</b>
<b>Appendix VII.3 Real-time File Handling</b>	
3.1 General overview	A VII.3-1
3.2 Application 1: Multiple Audio Channels	A VII.3-3
3.3 Application 2: Video Animation	A VII.3-5

This page is intentionally left blank

## CD-I Full Functional Specification

Appendix VII

CD-RTOS

### A VII.3 List of Illustrations

---

<b>Fig. No.</b>	<b>Caption</b>	<b>Page</b>
A VII.3.1a	Sector Layout 3 Audio Channels	A VII.3-3
A VII.3.1b	Sector Layout 3 Audio and 1 Video Channel	A VII.3-3
A VII.3.2	Real-time File Handling for Multiple Audio Channels	A VII.3-4
A VII.3.3	Sector Layout Video Animation	A VII.3-5
A VII.3.4	Real-time File Handling Video Animation	A VII.3-6

This page is intentionally left blank



## A VII.3 Real-time File Handling

---

### A VII.3 Real-time File Handling

#### 3.1 General Overview

A CD-I application is made up of a set of real time and non-real time segments. Typically, the non-real time segments are used for interaction with the user to determine the next real time segment to play, although a real time segment may also contain provisions for interaction.

In order to play a real time record on disc, an application must initialize several structures and set up an intercept routine to parse incoming signals. The signals sent to an application will include those indicating End-of-Record, End-of-File, Buffer Full and the presence of a trigger bit in a sector. These signals may be used to synchronize an audio/visual effect or to simply let the application know the state of the "play".

There are two primary structures that an application will need to set up and one connecting structure that ties the two together. The two primary structures are the Play Control Block and the Play Control List. There is only one Play Control Block for each Play operation. A Play Control List is created for each channel of Audio, Video, or Data sectors which will be transferred to memory. Audio sectors transferred directly to the Audio processor do not need an associated PCL. Since the PCB has only one field each for Audio, Data and Video sectors, the third structure, a CIL, is used to create an array of pointers to PCL's for the PCB. The channel number of the sector is used as an index into the CIL to find the appropriate PCL.

In general it is recommended to group non-real time sectors at the beginning of real time records. This type of data may be information about the file organization, CLUT values, premastered LCT's or program data. This type of data should be protected using mode 1 sectors with ECC. In addition, the data should be duplicated where possible.

Real time sectors contain Audio, Video and Program Data that are to be played back synchronously. Audio data can be output directly to the audio processor or can be transferred to memory. Video and program data are always transferred to memory.

### A VII.3 Real-time File Handling

---

Each sector is coded with a channel number that can be used to identify related sectors. There are 16 audio channels and 32 channels for video/program data. The application controls which sectors are accessed using two flags in the Play Control Block. PCB\_Chan is used to identify which channel numbers are significant. PCB\_AChan is used to identify which audio channels will be sent to the audio processor. Each of these flags is a bit-mapped value, i.e. each bit represents a channel.

In a typical application, a real time record would contain an audio sound track accompanying a series of images. When a video buffer is filled, the associated image is displayed and a second buffer is used to load the next image. We will now look closely at two specific applications which demonstrate some simple Real Time Record handling mechanisms.

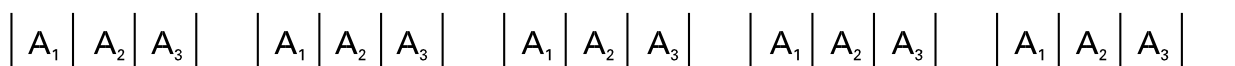
A VII.3 Real-time File Handling

---

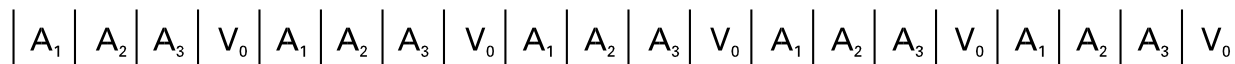
**3.2 Application 1: Multiple Audio Channels**

This application is a multi-lingual real time record. While images are viewed on the display, a sound track explains the images. At any time, the user may select one of three languages to listen to. The change should be instantaneous and seamless.

This is implemented using one video channel and three audio channels. The video data is placed in channel 0. Channels 1, 2 and 3 are used for audio data. Since audio data must appear at a constant rate, the layout of audio sectors is determined first. If B level stereo ADPCM data is used for all three sound tracks the initial sector layout would appear as follows:

Figure A VII.3.1a **Sector Layout 3 Audio Channels**

The video data would just have to fit in every fourth sector:

Figure A VII.3.1b **Sector Layout 3 Audio and 1 Video Channel**

This would allow loading of a full screen DYUV image every 2-2.5 seconds, depending on the image encoding format (PAL/NTSC). The formula for determining this is

$$((\text{width} * \text{height} * \text{pixel depth})/2324) * (13\text{ms} * 4).$$

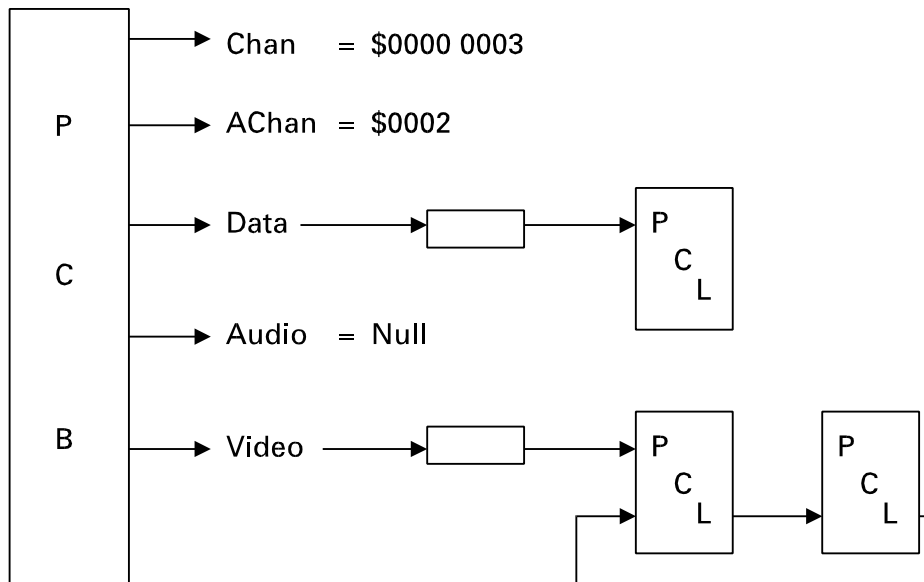
The Chan and AChan masks are set initially to transfer audio channel 1 to the audio processor. When the user wishes to change the language, the application sets the Chan and AChan masks to transfer either channel 2 or 3 to the audio processor. The video always stays in channel 0 and is always transferred to memory.

No PCL's are needed for the audio sectors since none are ever transferred to memory, but two will be used for the video data. They are set up in a circular list each pointing to the other, so that they will be accessed alternately. One will contain a pointer to a drawmap for plane A, and one will contain a pointer to a drawmap for plane B. As each buffer is filled, the application will get a signal indicating to display that image.

A VII.3 Real-time File Handling

---

Figure A VII.3.2 Real-time File Handling for Multiple Audio Channels



A VII.3 Real-time File Handling

---

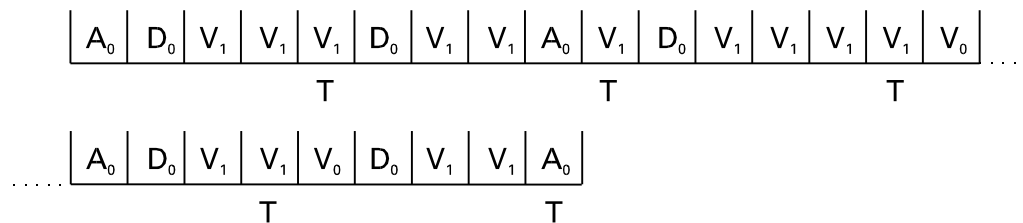
**3.3 Application 2: Video Animation**

The focus of this second application is video animation. There is a single audio channel and two interleaved video channels. One video channel contains DYUV images loaded into plane B as backgrounds. The second video channel contains runlength images which are updated rapidly in the foreground plane. If all of the runlength images use the same CLUT, then the CLUT would be placed in a Mode 2 Form 1 non-real time sector of the beginning of the Real Time Record. If a unique CLUT is needed for an image, it should be placed in a Mode 2 Form 1 real time data sector<sup>1</sup> immediately preceding the associated run length image video data.

In this example, there are two types of data which need to be spaced at specific intervals. The audio data will require regular spacing and the run length video data will need to be spaced evenly to give the animation a smooth flow.

If the sound quality level is B mono and the run length images were between 1 and 4 sectors in length, you might see a sector layout like this:

**Figure A VII.3.3 Sector Layout Video Animation**




---

<sup>1</sup> An eventual error in such a type of sector is not guaranteed to be corrected in real time by the CD-I system: the application must conceal it (e.g. ignore the corresponding picture).

Using instead a Mode 2 Form 1 non-real time sector would have caused a "hick-up" in case of error (due to the processing time of the ECC field): this can be acceptable in some circumstances.

Duplication is another technique to store such type of data within "real time" sectors.

A VII.3 Real-time File Handling

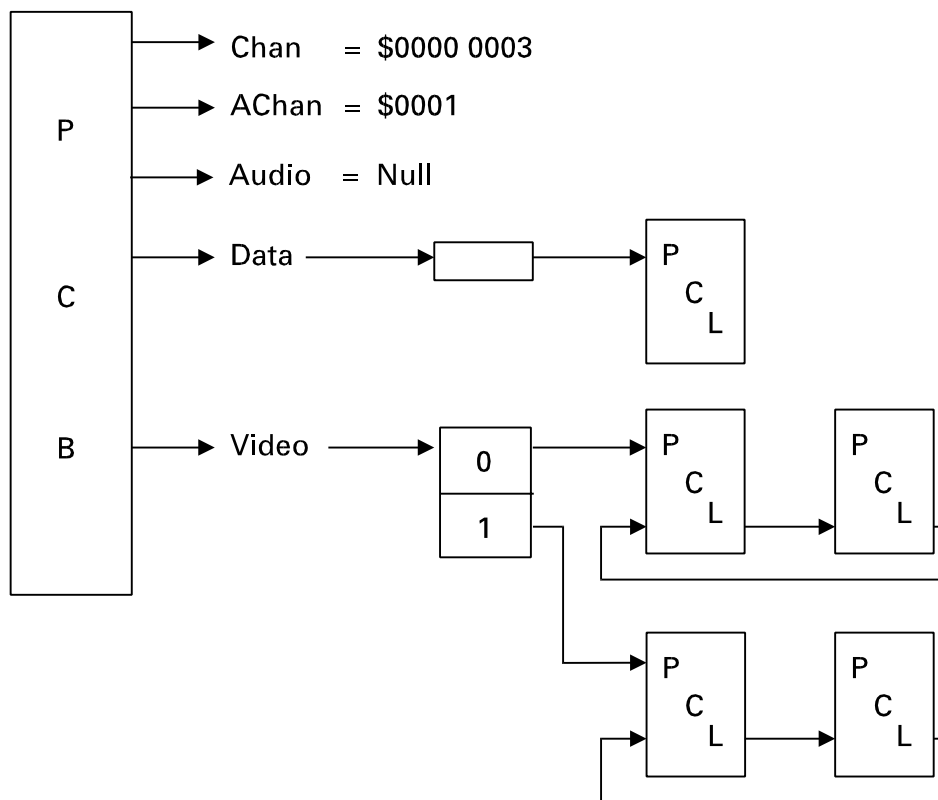
---

This allows the video to be updated at 15 frames per second. By placing a trigger bit in every 5th sector, the application can be notified when it is time to display a new image. By updating a smaller portion of the image or updating at a slower frame rate, the application can allow more room in the Real Time Record for background video data or larger run length images.

If a single runlength picture is greater in size than the allowed four sectors, it might have to be put into a different channel, thus allowing the application to put one or more of the sectors into an empty space where another image used less than 4 sectors.

The PCB\_Chan mask in this specific example would be set to access channels 0 and 1. AChan would reference channel 0.

Figure A VII.3.4 Real-time File Handling Video Animation



**Appendix VIII Base Case System Performance Figures**

This appendix is not yet completed

This page is intentionally left blank



Table of Contents

---

<b>Notes</b>	<b>Page</b>
Introduction	AN-v
Note 1 Base Case Performance Figures (Video)	AN 1-1
1.1 Introduction	AN 1-1
1.2 General Functions	AN 1-2
1.3 Drawmap Control Functions	AN 1-3
1.4 Regions	AN 1-4
1.5 Drawing Parameter Functions	AN 1-5
1.6 Graphics Drawing Functions	AN 1-6
1.7 Display Control Functions	AN 1-8
1.8 Video Inquire Functions	AN 1-9
1.9 Character Output Functions/Terminal Emulation	AN 1-10
Note 2 Base Case Performance Figures (Audio)	AN 2-1
2.1 Introduction	AN 2-1
2.2 General Audio Functions	AN 2-2
2.3 Getstat Functions	AN 2-3
2.4 Soundmap Functions	AN 2-4
2.5 Sound Control/Manipulation Functions	AN 2-5
2.6 Setstat Functions	AN 2-6
Note 3 List of Known Bugs for CD-RTOS v1.1	AN 3-1
3.1 Video Driver and Link in first LCT line	AN 3-1
3.2 Video Driver and execution of first LCT line	AN 3-2
3.3 Video Driver and LCT length	AN 3-3
3.4 CDFM and Sleeping Sickness	AN 3-4
3.5 CDFM and reading the raw device	AN 3-5
3.6 CDFM and Permission Testing	AN 3-6
3.7 Kernel and Memory Return	AN 3-7
3.8 SS_Play/SM_Out interaction	AN 3-8
3.9 Video Driver and DC_PWrlCT or DC_PRdLCT system call	AN 3-9
Note 4 CD Subcode Graphics for CD-I	AN 4-1
4.1 Introduction	AN 4-1
4.2 Extension System Call for CD-I	AN 4-3

This page is intentionally left blank

Table of Contents

---

<b>Notes</b>	<b>Page</b>
Note 5 Multi Disc Applications	AN 5-1
5.1 Introduction	AN 5-1
5.2 Caddy CD-I players	AN 5-2
5.3 Tray CD-I players	AN 5-2
5.4 Multi disc changers	AN 5-3
5.6 Conclusion	AN 5-3

This page is intentionally left blank

Notes in this Addendum provide additional explanations, interpretation guidelines and further information to the "CD-I Full Functional Specification".

This page is intentionally left blank

1.1 Introduction

---

**NOTE 1: Base Case Performance Figures (Video)**

**1.1 Introduction**

This note contains the results of performance measurements for a selection of UCM video functions.

Performance figures are measured with the CD-RTOS 1.1 Base-Case Validation suite on Disc 1 and given as typical process times, where process time is the pure execution time of the C-binding of a UCM function. Only typical values are shown. The values can vary according to the implementation of the particular player used. However, these values should only be used as a guideline. Times are given in milliseconds, rounded off upwards to three significant digits.

The UCM functions are conforming to release 1.1 of the CD-I Full Functional Specification, dated September 1990.

Performance figures were measured for a base-case CD-I player with the following characteristics:

- \* Philips CD-I player, model 605 (reference or "Sanko" player)
- \* 68070 processor
- \* 1 Mbyte RAM
- \* CD-RTOS 1.1 (in ROM)

1.2 General Functions

---

**1.2 General Functions**

Function	Main Parameters <sup>(1)</sup>	Data Type	T (ms)
I\$Open	Access mode Read, Write	<sup>(2)</sup>	19.2
I\$Close		<sup>(2)</sup>	7.95

---

<sup>1</sup> The main parameters apply to all data types

<sup>2</sup> not applicable



### 1.3 Drawmap Control Functions

---

#### 1.3 Drawmap Control Functions

Function	Main Parameters	Data Type	T (ms)
DM_Create	Size W,H : 768,560	CLUT4	6.39
	768,560	CLUT4HiRes	7.10
	768,560	CLUT7	6.50
	768,560	CLUT8	6.50
	768,560	RGB555	9.30
DM_Exch	Size W,H : 768,464	CLUT4	245
	768,464	CLUT4HiRes	488
	768,464	CLUT7	245
	768,464	CLUT8	245
	768,464	RGB555	504
DM_Tcpy	Size W,H : 768,464	CLUT4	6669
	768,464	CLUT4HiRes	13350
	768,464	CLUT7	534
	768,464	CLUT8	532
	768,464	RGB555	920
DM_TExc	Size W,H : 768,464	CLUT4	9729
	768,464	CLUT4HiRes	19450
	768,464	CLUT7	857
	768,464	CLUT8	857
	768,464	RGB555	1540
DM_Write	Size W,H : 768,116	CLUT4	108
	768, 58	CLUT4HiRes	108
	768,116	CLUT7	28.5
	768,116	CLUT8	28.5
	768, 58	RGB555	95.7
DM_IrWr	Start vert., # lines	CLUT4	584
	0,116	CLUT4HiRes	583
	0, 58	CLUT7	183
	0,116	CLUT8	183
	0,116	RGB555	147
DM_Read	Size W,H : 768,116	CLUT4	109
	768, 58	CLUT4HiRes	108
	768,116	CLUT7	29.0
	768,116	CLUT8	28.5
	768, 58	RGB555	91.0
DM_Close	-	CLUT4	3.17
	-	CLUT4HiRes	3.10
	-	CLUT7	3.10
	-	CLUT8	3.10
	-	RGB555	4.90

1.4 Regions

---

**1.4 Regions**

Function	Main Parameters <sup>(1)</sup>	Data Type	T (ms)
RG_Create	Type : circle Center : x,y 360,240 Radius : 40	(4)	31.4
RG_Isect	Type 1 : circle <sup>(2)</sup> Type 2 : circle <sup>(3)</sup>	(4)	26.0
RG_Union	Type 1 : circle <sup>(2)</sup> Type 2 : circle <sup>(2)</sup>	(4)	36.0
RG_Diff	Type 1 : circle <sup>(2)</sup> Type 2 : circle <sup>(3)</sup>	(4)	31.3
RG_XOR	Type 1 : circle <sup>(2)</sup> Type 2 : circle <sup>(3)</sup>	(4)	47.0
RG-Move	Type : circle <sup>(2)</sup> To x,y: x+6, y+4	(4)	2.15
RG-Del		(4)	1.90

---

<sup>1</sup> The main parameters apply to all data types

<sup>2</sup> Circle with center x,y 200,200, radius 40

<sup>3</sup> Circle with center x,y 260,200, radius 40

<sup>4</sup> not applicable

## 1.5 Drawing Parameter Functions

---

### 1.5 Drawing Parameter Functions

Function	Main Parameters <sup>(1)</sup>	Data Type	T (ms)
DT_Ptn <sup>(2)</sup>		CLUT4 CLUT4HiRes CLUT7 CLUT8 RGB555	2.85 2.90 3.50 3.45 4.65
DP_PAln	Pattern Alignment: H,V 10,10	CLUT4 CLUT4HiRes CLUT7 CLUT8 RGB555	0.72 0.70 0.72 0.72 0.72
DP_SCMM	Map Method 0	CLUT4 CLUT4HiRes CLUT7 CLUT8 RGB555	0.70 0.70 0.70 0.72 0.70
DP_SCR	Color data: \$AABBCCDD	CLUT4 CLUT4HiRes CLUT7 CLUT8 RGB555	0.82 0.82 0.79 0.80 0.78
DP_GFnt <sup>(3)</sup>	Font name: font8x8	CLUT4 CLUT4HiRes CLUT7 CLUT8 RGB555	2.09 5.60 3.50 4.10 4.70
DP_Afnt <sup>(4)</sup>		CLUT4 CLUT4HiRes CLUT7 CLUT8 RGB555	2.46 0.78 0.75 0.75 0.77
DP_DFnt		CLUT4 CLUT4HiRes CLUT7 CLUT8 RGB555	2.41 2.50 0.67 0.67 0.70
DP_Clip		CLUT4 CLUT4HiRes CLUT7 CLUT8 RGB555	3.15 3.50 3.35 3.35 3.30

---

<sup>1</sup> The main parameters apply to all data types

<sup>2</sup> Pattern type is of given type

<sup>3</sup> Assumes font8\*8 is in memory

<sup>4</sup> Active font is 0

## 1.6 Graphics Drawing Functions

## 1.6 Graphics Drawing Functions

Function	Main Parameters <sup>(1)</sup>	Data Type	T (ms)
DR_Rect	Init. corner H,V 40, 100 Oppos. corner 90, 150 Filled	CLUT4	14.7
		CLUT4HiRes	27.7
		CLUT7	14.7
		CLUT8	14.7
		RGB555	16.0
DR_Erect	Init. corner H,V 40, 100 Oppos corner 90, 150 Radii curv. H,V 20, 10 Filled	CLUT4	35.0
		CLUT4HiRes	46.5
		CLUT7	24.0
		CLUT8	24.0
		RGB555	27.0
DR_PGon	Vertices: 10 600,50/500,150/600,250 500,350/600,450/100 450/200,350/100,250 200,150/100,50 Outlined	CLUT4	374
		CLUT4HiRes	375
		CLUT7	186
		CLUT8	186
		RGB555	202
DR_Elps	Center point H,V: 360, 240 Radii H,V: 35, 25 Filled	CLUT4	44.5
		CLUT4HiRes	51.0
		CLUT7	26.0
		CLUT8	26.0
		RGB555	28.0
DR_EWdg	Start angle H,V 600,100 End angle H,V 40,400 Center H,V 40,100 Radii H,V 50,25 Outlined	CLUT4	40.0
		CLUT4HiRes	45.0
		CLUT7	28.5
		CLUT8	28.4
		RGB555	29.5
DR_gn	Rectangle region Size (H,V), (H,V) 40,100 140,200 Filled	CLUT4	28.5
		CLUT4HiRes	55.5
		CLUT7	28.5
		CLUT8	28.5
		RGB555	32.0

continued / ...

## 1.6 Graphics Drawing Functions

---

### Graphics Drawing Functions (continued)

Function	Main Parameters <sup>(1)</sup>	Data Type	T (ms)
DR_BFill <sup>(2)</sup>	Drawmap W,H 768,560 Init. coords. 240,300 Bound. color 0	CLUT4 CLUT4HiRes CLUT7 CLUT8 RGB555	1150 2270 460 460 550
DR_FFill <sup>(3)</sup>	Drawmap W,H 768,560 Init. coords. 240,300	CLUT4 CLUT4HiRes CLUT7 CLUT8 RGB555	9.64 10.0 10.0 10.0 9.50
DR_Text <sup>(4)</sup>	Max. no. chars. 25 Position H,V 40,100	CLUT4 CLUT4HiRes CLUT7 CLUT8 RGB555	63.9 64.0 57.0 57.0 66.0

---

<sup>1</sup> The main parameters apply to all data types

<sup>2</sup> Rectangle x,y 40-440,100-500

<sup>3</sup> Rectangle x,y 40-640,100-500

<sup>4</sup> string "THIS IS A STRING"

1.7 Display Control Functions

---

**1.7 Display Control Functions**

Function	Main Parameters <sup>(1)</sup>	Data Type	T (ms)
DC_CrFCT	FCT length : 200	HiRes	4.20 3.20
DC_DIFCT		HiRes	2.00 1.90
DC_CrLCT	LCT length : 560	HiRes LoRes	41.3 21.3
DC_DILCT		HiRes LoRes	1.25 1.25
DC_WrLCT	No. of lines : 560 No. of columns : 1	HiRes LoRes	6.70 6.70
DC_FLnk		HiRes LoRes	0.85 0.85
DC_Exe		HiRes LoRes	0.70 0.70

---

<sup>1</sup> The main parameters apply to all data types

1.8 Video Inquire Functions

---

**1.8 Video Inquire Functions**

Function	Main Parameters <sup>(1)</sup>	Data Type	T (ms)
VIQ_JCPs	No. of text chars. : 25 Justification length : 25	CLUT4 CLUT4HiRes CLUT7 CLUT8 RGB555	3.69 3.70 3.90 3.90 3.90

---

<sup>1</sup> The main parameters apply to all data types

## 1.9 Character Output Functions/Terminal Emulation

---

### 1.9 Character Output Functions/Terminal Emulation

Function	Main Parameters <sup>(1)</sup>	Data Type	T (ms)
I\$_Write	font8x8 100 bytes	CLUT4	160
		CLUT4HiRes	161
		CLUT7	60.5
		CLUT8	60.5
		RGB555	75.9
Cursor-up	200 lines	CLUT4	0.85
		CLUT4HiRes	0.85
		CLUT7	0.85
		CLUT8	0.85
		RGB555	0.85
Clear-to-EOS	Entire screen	CLUT4	123
		CLUT4HiRes	245
		CLUT7	123
		CLUT8	122
		RGB555	222
Delete-char	Home position	CLUT4	6.19
		CLUT4HiRes	6.30
		CLUT7	5.90
		CLUT8	5.90
		RGB555	9.10
Insert-char	Home position No. of columns : 1	CLUT4	6.29
		CLUT4HiRes	6.40
		CLUT7	5.90
		CLUT8	6.00
		RGB555	9.30
Insert-line	First screen line	CLUT4	115
		CLUT4HiRes	227
		CLUT7	115
		CLUT8	115
		RGB555	230

---

<sup>1</sup> The main parameters apply to all data types



## 2.1 Introduction

---

### **NOTE 2: Base Case Performance Figures (Audio)**

#### **2.1 Introduction**

This report contains the results of performance measurements for a selection of CDFM audio functions.

Performance figures are measured with the CD-RTOS 1.1 Base-Case Validation Suite on Disc 2 and given as typical process times, where process time is the pure execution time of the C-binding of a CDFM function. Only typical values are shown. The values can vary according to the implementation of the particular player used. However, these values should only be used as a guideline. Times are given in milliseconds, rounded off upwards to three significant digits.

The CDFM functions conform to release 1.1 of the CD-I Full Functional Specification, dated September 1990.

Performance figures were measured for a base-case CD-I player with the following characteristics:

- \* Philips CD-I player, model 605 (reference or "Sanko" player)
- \* 68070 processor
- \* 1 MByte RAM
- \* CD-RTOS 1.1 (in ROM)

2.2 General Audio Functions

---

**2.2 General Audio Functions**

Function	Main Parameters	P (ms)	T (ms)
I\$Open	CDDA device	19.6	12.1
	CDDA file	19.6	1490
	CDI	19.6	2380
I\$Close	CDDA device	0	4.00
	CDDA file	0	4.13
	CDI file	0	4.00

2.3 Getstat Functions

---

**2.3 Getstat Functions**

Function	Main Parameters	P (ms)	E (ms)
SS_EOF		0.46	0.46
SS_CDFD		0	0.68
SS_Opt		1.29	0.66
SS_Path		0	0.73
SS_Pos		2.96	0.51
SS_Size		1.29	0.54

2.4 Soundmap Functions

---

**2.4 Soundmap Functions**

Function	Main Parameters	P (ms)	E (ms)
SM_Close	1 sector soundmap	2.87	1.84
	150 sector soundmap	1.26	1.81
SM_Cncl	150 sector no errors	0.70	0.71
	150 sector 10 errors	3.10	3.26
SM_Creat	1 sector soundmap	4.62	4.96
	150 sector soundmap	315	316
SM_Info		0	0.61
SM_Off		2.96	3.62
SM_Out	first for mixing	4.59	3.50
SM_Stat	after a change	3.45	0.69

2.5 Sound Control/Manipulation Functions

---

**2.5 Sound Control/Manipulation Functions**

Function	Main Parameters	P (m)	E (ms)
SC_Atten		0	0.78
SD_Loop	loop 1 loop 2	2.93 0	0.87 0.86
SD_MMix	A mono mix B/C mono mix	10.1 15.8	10.2 15.8
SD_SMix	B/C mixLL B/C mixLR B/C mixRL B/C mixRR	6.71 5.83 8.69 7.51	6.83 5.90 8.05 7.06

## 2.6 Setstat Functions

## 2.6 Setstat Functions

Function	Main Parameters	P(ms)	E(ms)
SS_Seek	full	2210	2210
	CDDA file	1950	1960
	CDI file	1840	1840
SS_Abort	CDDA device	0	1.88
	CDDA file	0	1.88
	CDI file	9.60	1.80
SS_CChan		0	17.9
SS_CDDA	device	9.60	2.75
	file	0	2.73
SS_CONT	CDDA device	0	2.63
	CDDA file	0	2.52
	CDI audio	0	289
	CDI data	0	211
SS_Disable		0	0.64
SS_Eject		9.60	27.9
SS_Enable		0	0.79
SS_Mount		0	0.64
SS_Opt		0	0.48
SS_Pause	CDDA device	0	21.8
	CDDA file	0	25.1
	CDI audio	0	26.4
	CDI data	0	21.8
SS_Play	()	9.60	2.61
	KIT02_A	0	2.89
	KIT02_B	9.60	2.83
SS_Raw	device	9.60	1430
	File 1 sector	9.60	1470
	File 150 sectors	9.60	2080
SS_Readtoc		0	1950
SS_Seek	device LSN#0	0	2.76
SS_Stop	CDDA device	9.60	19.7
	CDDA file	0	19.9
	CDI file	0	38.9

3.1 Video Driver and link in first LCT line

---

**NOTE 3: Known Bug List for CD-RTOS v1.1**

**3.1 Video Driver and Link in first LCT line**

**Problem description**

If you put an LCT Link in the first line of an LCT that is linked to an FCT, the FCT becomes corrupted.

**Explanation**

Normally a link instruction in an FCT or LCT is translated into some "hidden" instructions like:

First line of LCT	reload DCP	reload DLSP + stop	reload DCP + stop
-------------------	------------	--------------------	-------------------

If the last instruction of the first line in the LCT is an instruction linking this LCT line to another LCT, the translation into these "hidden" instructions is not performed correctly.

Effectively, this means that the reload DLSP instruction is never seen.

**Work around**

Do not write a link instruction into the first line of the LCT linked to by the FCT.

### 3.2 Video Driver and execution of first LCT line

---

#### 3.2 Video Driver and execution of first LCT line

##### Problem description

It is possible that the first line of an LCT is executed BEFORE the final instruction of the FCT for the opposite plane.

##### Explanation

Page VII-81 states that "the two DCPs are executed concurrently by the display control hardware." Page VII-77 states that "Interpretation of the FCT is completed before the start of the active display period". Page VII-78 states that "Immediately prior to the display of the first line of pixels, the display control hardware will interpret the instructions contained in the line of the LCT pointed to by this link."

This implies that the FCTs are executed completely (and concurrently) before the LCTs are executed. Then each line of the two LCTs is executed simultaneously.

This is, in fact, not the case.

In the current implementation of the video driver, the processing of Plane A's FCT is slightly delayed from the processing of Plane B's FCT, This is due to the processing of the Graphic Cursor Instructions that are a "hidden" part of Plane A's FCT. Secondly, the first line of the LCT linked to by the FCT is actually copied into and executed from the FCT. Because of this, and the fact that the FCTs may be of different length, it is possible that the end of Plane A's FCT is actually executed AFTER the first line of plane B's LCT (or vice versa).

The obvious implication is that instructions which set a condition in line 0 of a given Plane's LCT could be reset by the other Plane's FCT.

##### Work around

Do not set instructions in line 0 of one Plane's LCT that are also being set by the other Plane's FCT or LCT.



### 3.3 Video Driver and LCT length

---

#### 3.3 Video Driver and LCT length

##### **Problem description**

The maximum length of an LCT is 2048 lines (0-2047).

##### **Explanation**

Page VII-77 indicates that an LCT may have up to 65535 lines. However, a bug in the video driver does not allow you to write to a line number greater than 2047.

##### **Work around**

Do not create an LCT longer than 2048 lines (0-2047).

### 3.4 CDFM and Sleeping Sickness

---

#### 3.4 CDFM and Sleeping Sickness

##### **Problem description**

When there are several (3 or more) processes trying to access the disc concurrently, it is possible that one of them may go to sleep and not wake up.

##### **Explanation**

If you start up 3 or 4 concurrent processes each executing "dir -er /cd" (print an extended directory listing of all files on the CD) occasionally one of the processes will go to sleep and never wake up.

##### **Work around**

None

### 3.5 CDFM and reading the raw device

---

#### 3.5 CDFM and reading the raw device

##### Problem description

Opening the disc for raw I/O (using the "/cd@" mechanism) does not allow a process to access sectors with non-zero file numbers. Additionally, it will sometimes stop I/O operations based on seeing an EOF bit.

##### Explanation

Page VII-71 states that opening a CD for raw I/O "allows the process to access the information on the device independently of the file structure." In fact, you can only read sectors that have "File Number 0".

Secondly, EOF bits are sometimes acknowledged as well. For example, the SS\_Play operation will terminate at EOF. Because I\$Read commands are internally converted to SS\_Play commands this means that I\$Read will actually terminate in sectors containing EOF bits. If you read sectors one at a time, this is no problem. If you are reading multiple sectors and you encounter an EOF, I\$Read will return only sectors up to the one containing EOF, but it will indicate that the read was completed without error.

##### Work around

There is no mechanism for accessing every sector on the disc when you are using Interleaved Files (sectors with non-zero file numbers). This part of the bug has no work around.

For the EOF problem, you must read only single sector blocks or be aware of file boundaries.

### 3.6 CDFM and Permission Testing

---

#### 3.6 CDFM and Permission Testing

##### Problem description

When CDFM checks a process's user ID to validate it for SS\_Abort and SS\_Pause, it is compared to the owner of the path, not the user that actually started the play operation.

##### Explanation

The SS\_Pause and SS\_Abort functional descriptions state that they "may be used only by a process with a user ID of the super user or with the user ID of the process which stated the operation." In fact, the user ID of the process that executed the \$Open call, not the SS\_Play, SS\_Seek, SS\_CDDA operations.

Therefore the following sequence would cause SS\_Pause to return an error:

```
setuid(2.1)
open(/cd)
setuid(3.1)
SS_Play()
SS_Pause()
```

On the other hand, this is somewhat improved by the fact that all CD-I applications are forked initially as user 0.0 which has no restrictions.

##### Work around

Do not change your process user ID between the time that you open a path to a file and the time that you start (or stop) an asynchronous operation on that path.

### 3.7 Kernel and Memory Return

---

#### 3.7 Kernel and Memory Return

##### **Problem description**

The system call F\$SRtMem can return error #237 even though it does return the memory.

##### **Explanation**

If every byte of memory on the CD-I system is allocated and then deallocated, the memory deallocations (F\$SRtMem) will return error #237 (E\$MemFul), although they will actually return the memory.

##### **Work around**

Ignore error #237 on F\$SRtMem.

### 3.8 SS\_Play/SM\_Out interaction

---

#### 3.8 SS\_Play/SM\_Out interaction

##### **Problem description**

When executing SS\_Play and SM\_Out concurrently, it is possible for audio sectors on the disc to be transferred to memory when they would normally be going to the audio processor.

##### **Explanation**

If an SS\_Play is active that is streaming audio sectors to the audio processor, and a process executes SM\_Out the AChan mask in the CDC is temporarily set to 0 (do not transfer sectors from disc to AP). During this time these audio sectors are subject to normal rules for sector selection using the PCB\_Chan, CIL and PCB mechanisms.

Often this will not be noticed: when a process asks audio sectors to be streamed to the AP there will not be a PCL set up to receive sectors on that audio channel. If there is a PCL set up, though, it will receive the sectors until the soundmap is done playing.

##### **Work around**

If you do not wish to receive audio sectors from the channel specified by PCB\_AChan during a concurrent SM\_Out, write a NULL into the appropriate entry in the Audio CIL.

If you want to receive audio sectors from the channel specified by PCB\_AChan during a concurrent SM\_Out then this work around will not work.

3.9 Video Driver and DC\_PWrLCT or DC\_dLCT system call

---

**3.9 Video Driver and DC\_PWrLCT or DC\_PRdLCT system call**

**Problem description**

Using the DC\_PWrLCT and DC\_PRdLCT system calls, you cannot access a line greater than or equal to 512. Using the DC\_WrLCT and DC\_RdLCT system calls, you cannot access a line greater than or equal to 1024.

**Explanation**

In the driver, the line number is multiplied times 64 to calculate the byte offset of the start of that line. With numbers greater than or equal to those listed above, this offset becomes greater than 0x8000. When this offset is added to the LCT start address, a "signed" addition is used, causing the offset to appear as a negative number. Thus, the wrong offset is generated and the instruction is written into an incorrect area. This may corrupt other memory.

**Work around**

None

This page is intentionally left blank



## 4.1 Introduction

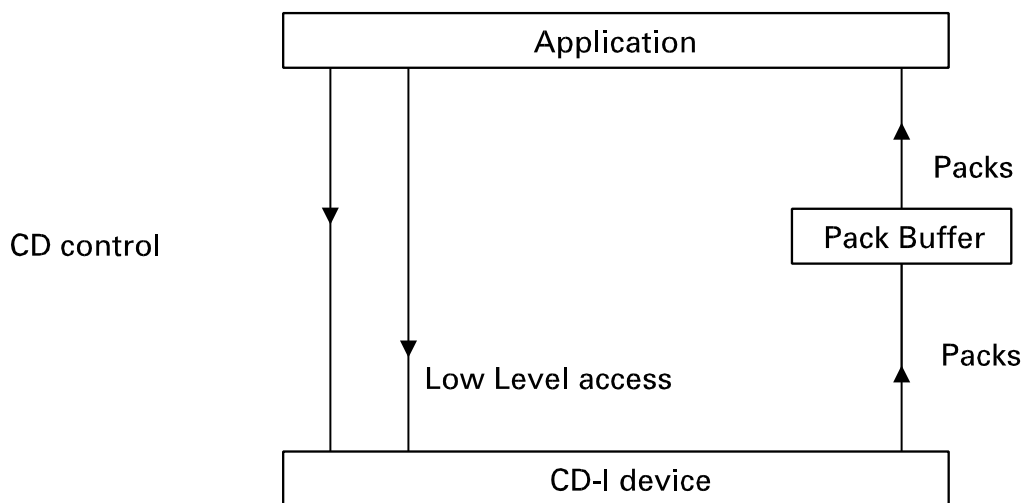
**NOTE 4: CD Subcode Graphics for CD-I****4.1 Introduction**

This addendum describes the CD-I extension: CD Subcode Graphics.

Subcode Graphics is defined in the System Description Compact Disc Digital Audio. The same naming convention is used in this addendum.

The call described delivers de-interleaved, error corrected packs in a pack buffer during the execution of the SS\_CDDA function. The application is responsible for emptying the pack buffer and further handling of the packs. The application is signalled whether and how many packs are stored in the buffer.

The play will abort when the buffer overflows. Low level access is accomplished by using the functions SS\_CDDA (Base Case CD-I function) and SS\_Subon (Extended CD-I function).

**Pack Buffer**

The pack buffer is a circular FIFO buffer. The size of the pack buffer is a multiple of the pack size and must be able to hold at least 4 packs. A buffer location contains a pack when the two most significant bits of the pack header are set. After being processed the application must reset these bits to indicate the location is free again. A buffer overflow will abort the play.

## 4.1 Introduction

---

### Pack Structure

The pack being delivered by the CD device consists of 20 symbols. Each symbol is stored in a byte. The content of a pack is defined as:

symbol 0 MODE, ITEM  
 symbol 1 COMMAND  
 symbol 2 STATUS  
 symbol 4 DATA WORD 0

·  
 ·  
 symbol 19 DATA WORD 15

The STATUS symbol is defined:

(msb)	5	4	3	2	1	0	(lsb)
	0	Q	E	P	0	0	

Q = 1      Uncorrectable Q parity error (Q>1).  
 E = 1      Invalid data (e.g. first 7 packs).  
 P = 1      Uncorrectable P parity error (P>2).

For all symbols only bits 0-5 contain valid information (see CD Subcode Graphics description). In symbol 0 the 2 msb's are used to indicate whether the pack buffer location contains a pack or not.

## 4.2 Extension System Call for CD-I

---

### 4.2 Extension System Call for CD-I

#### SS\_Subon - Enable subcode retrieval

This function enables the retrieval of subcode information during the execution of SS\_CDDA. The retrieval function is automatically disabled at the end of the play. SS\_Subon causes the de-interleaved error corrected packs of subcode data to be stored in the (circular FIFO) pack buffer. The size of the pack buffer must be a multiple of 20 bytes, being the pack size, and must be at least 80 bytes.

The msb and msb-1 of the first byte of the pack in the buffer are set to indicate the buffer contains a pack to be processed by the application. After processing the application must clear these bits. When the system delivers pack(s) into the buffer, the application defined event is signalled indicating the amount of packs stored. The system will abort an ongoing play when a buffer overflow is detected. The application is informed via the STAT\_BLK of the SS\_CDDA function call.

The mode/item structure can be modified by the application during the play.

It is recommended to call SS\_Subon directly before SS\_CDDA.

Input:

d0.w	=	path number
d1.w	=	SS_Subon setstat code
d2.l	=	ID of event
d3.l	=	size of pack buffer
(a0)	=	pointer to pack buffer
(a1)	=	pointer to mode/item structure

Output: None

Error output:

cc	=	carry bit set to one
d1.w	=	error code

Possible errors: E\$UnkSvc, E\$EvtID

4.2 Extension System Call for CD-I

---

The definition of the fields in the mode item structure are:

Reserved					on/off
Reserved			moem		
ch 15	-----	ch 8	ch 7	-----	ch 0
bit 15	-----	bit 8	bit 7	-----	bit 0

on/off: 0

**TRANSPARENT MODE:**

all retrieved subcode packs are stored in the buffer. The content of the moem and the channel field are ignored. During the subsequent play these fields reflect the mode, item and channel of the last returned pack.

1

**SELECTIVE MODE:**

only packs of the type as indicated by the moem field are stored in the memory.

moem: 00000000  
 00001000  
 00001001  
 00001010  
 00011000  
 00111000

Zero mode  
 Line graphics mode  
 TV graphics mode  
 Extended TV graphics (includes TV Graphics)  
 Midi mode  
 User mode

Bit 0..2 select the item, bits 3..5 select the mode and bits 6..7 must be zero.

channel:

each bit represents a channel (0-15). All packs within the channels of which the corresponding bit is set are stored in the pack buffer. The channel field is only taken into account in the TV and Extended TV Graphics mode.

## **NOTE 5: Multi Disc Applications**

### **5.1 Introduction**

This note explains how to prepare multi disc applications for CD-I players.

Because of the mechanical differences between CD-I players with a caddy and with a tray loading mechanism, the behaviour of these players will be slightly different for multi disc applications.

## 5.2 Caddy CD-I players

---

### 5.2 Caddy CD-I players

Removing the optical disc by inserting the empty caddy housing when the CD-I application is running can cause a system (hardware) reset in the player. To prevent this system reset the CD-I application should execute an "SS\_Eject" call (Issue Door Open Command to Disc Drive, VII.2.2.3.2). Once this call is executed, the player will not reset any more when new discs are inserted until it recognizes a valid disc type. The newly inserted disc can now be used again by the (still running) CD-I application. Be aware that once a valid type has been recognized, the removal of this disc will again cause a system reset.

**Note:** Using the "SS\_Disable" (Disable Hardware Control Buttons on Player, page VII.2.2.3.2) disables only the player control keys.

### 5.3 Tray CD-I players

Removing the disc by pressing Open/Close when the CD-I application is running can cause a system (hardware) reset in the player. When the CD-I application is running, the Open/Close button can act as a combined Reset-Open/Close button. For Multi Disc CD-I applications one should always use the "SS\_Eject" call (Issue Door Open Command to Disc Drive, VII.2.2.3.2). This call will open the tray without resetting the player. If the new disc is inserted, the tray will again be closed by using the "SS\_Mount" call (Mount disc by disc number, VII.2.2.3.2) or by any other call that accesses the disc.

**Note:** Using the "SS\_Disable" (Disable Hardware Control Buttons on Player, VII.2.2.3.2) disables only the player control keys (except the Open/Close).

## 5.4 Multi disc changers

---

### 5.4 Multi disc changers

For multi disc changers the implementation is as follows:

The "SS\_Eject" call (Issue Door Open Command to Disc Drive, VII.2.2.3.2) must be used to open the multi disc loader without resetting the player, and give the user the possibility to change discs.

The "SS\_Mount" call (Mount disc by disc number, VII.2.2.3.2) will mount a disc on a location in the loader specified by the disc number. The loader will always be closed (if the loader was open it will be closed again). An SS\_Mount(0) will cause the same disc to be loaded. SS\_Mount(1) will cause the next disc to be loaded. SS\_Mount(4) will cause the previous disc to be loaded (in the case of a 5 disc changer). It is the application's responsibility to find the next disc in the loader. The discs will, in general, not be available in the proper order from the loading mechanism.

So for multi disc applications in a changer, the application should only use the SS\_Mount call to get to the next disc. If it does not find the next disc in the loader, it can perform an SS\_Eject to open the loader and ask the user to change or insert the wanted disc.

### 5.5 Conclusion

A CD-I application must take into account whether it runs on a single or a multi disc player. This can be detected by reading the CSD file. Device Type Code 6 (CD Player, Appendix page A VII.2.2.6.3) will inform the application whether the player is a single or a multi disc type. When it has detected the player type it can act in an appropriate way, handling multiple disc applications.

This page is intentionally left blank